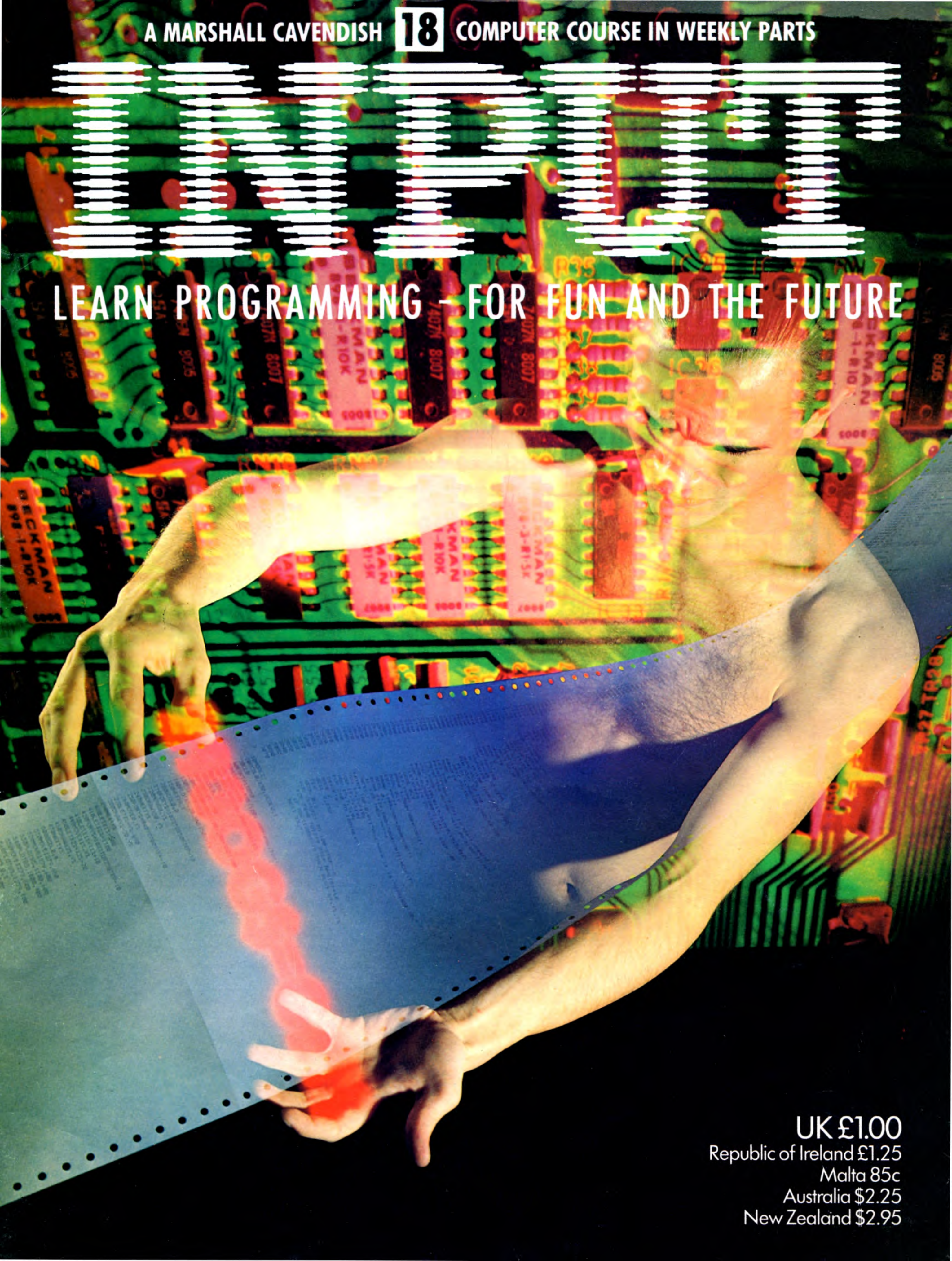
A MARSHALL CAVENDISH 18 COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95

INPUT

Vol. 2 **No 18**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS
Front cover, Graeme Harris. Pages 541, 542, 544, Artist Partners/Tony Healey. Pages 546, 548, 550, Graeme Harris. Pages 553, 554, 556, 559, Gary Wing. Page 560, Projection/Trevor Lawrence. Pages 562, 563, Bernard Fallon. Page 565, Kevin O'Keefe. Pages 566, 571, Malcolm Livingstone. Pages 568, 569, Ray Duns.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland:
Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

WHO NEEDS WORDPROCESSING

- ELECTRONIC CORRECTIONS
- CLEAN TEXT MANIPULATION
- HARDWARE NEEDS
- PRINTOUT—THE END RESULT



Wordprocessing is an elegant, almost magic twentieth century solution to many of the problems which have plagued wordsmiths since writing was invented

Painting, carving, quilling and—in present day context—penning or typing is fine only if you're sure you can get your work absolutely letter perfect the first time around.

If, like most of us, you make mistakes—or simply change your mind about what you want—corrections and changes have to be made. Not only do these take time—they nearly always look messy. And this is no good if you're out to impress somebody with your letter, report or article.

If these changes and corrections are numerous or extensive, then you'll have to rewrite or retype large chunks of the document—perhaps even all of it. And there's a fair chance you'll make other mistakes the second time around.







These three types of screen display are commonly used on wp programs mainly intended for home use

One of the significant, immediate benefits of wordprocessing is that it allows you to eliminate your mistakes easily and without having to rework all the good stuff as well. And that can mean tremendous time savings even to people who may only occasionally have cause to put pen to paper.

GOODBYE CORRECTING FLUID

So what is wordprocessing? Well, instead of typing your words directly onto a sheet of paper—where they are difficult and messy to change—you type them first into a computer's memory. The words can easily and neatly be changed while they are stored here.

At the very simplest level, spelling mistakes or other 'literals' can be corrected simply by moving your electronic 'nib'—a cursor—to a suitable point and retyping. The error is erased from memory and replaced with the correct version by overwriting. No mess, no fuss ... and you can make the correction whenever you like, before printing the document on a printer connected to your computer.

If you want to insert words in the middle of a paragraph, you can do so. One way is to insert the necessary number of spaces and then type in the insert copy. Another is to let the machine automatically shuffle the rest of the text further along to make room for the inclusion.

These are just two of the commonly found features—the real strength of wordprocessing really only becomes evident when you consider all the rather special electronic tricks you can use in your writing.

MULTIPLE LETTERS

For instance, you can send essentially the same letter to any number of different people, simply by reprinting it. The quality of the printed letter (although this obviously depends on your printer) would be very much better than a photostat of the original.

You can go a stage further by overwriting parts of a *standard letter* with a new name and address panel, references, salutation—and date. With these it is a simple matter to personalise them even more by inserting a paragraph, or whatever, of extra writing. Any number of stock letters can be kept 'on file' for use in this way, including both personal and business correspondence.

One stage further is to use a *form letter* which lets the computer do all the work once you've keyed in what's called a *fill file*, which is really only a small datafile.

A form letter is a standard letter in which blanks or *blocks* are left at every point where new data is to be inserted to complete the letter. So, typically, you'd have one block for the name, one for each line of the address, one for a reference, one for the salutation (the bit after the 'Dear'), and one for each point in the letter where you might want to change from letter to letter. For invoices or statements, you would need blocks relating to payments owing or made.

The fill file is then completed. For most purposes this, too, is created on the word processor and consists only of lines corresponding to the blocks (the 'blanks') of the form letter. If you have five blocks you need five lines or entries of fill copy—no more, no less. One set of block fill copy is required for each letter.

THE PRINT OUT

Then comes the interesting bit. You set up the wordprocessor to print the form letter. Each

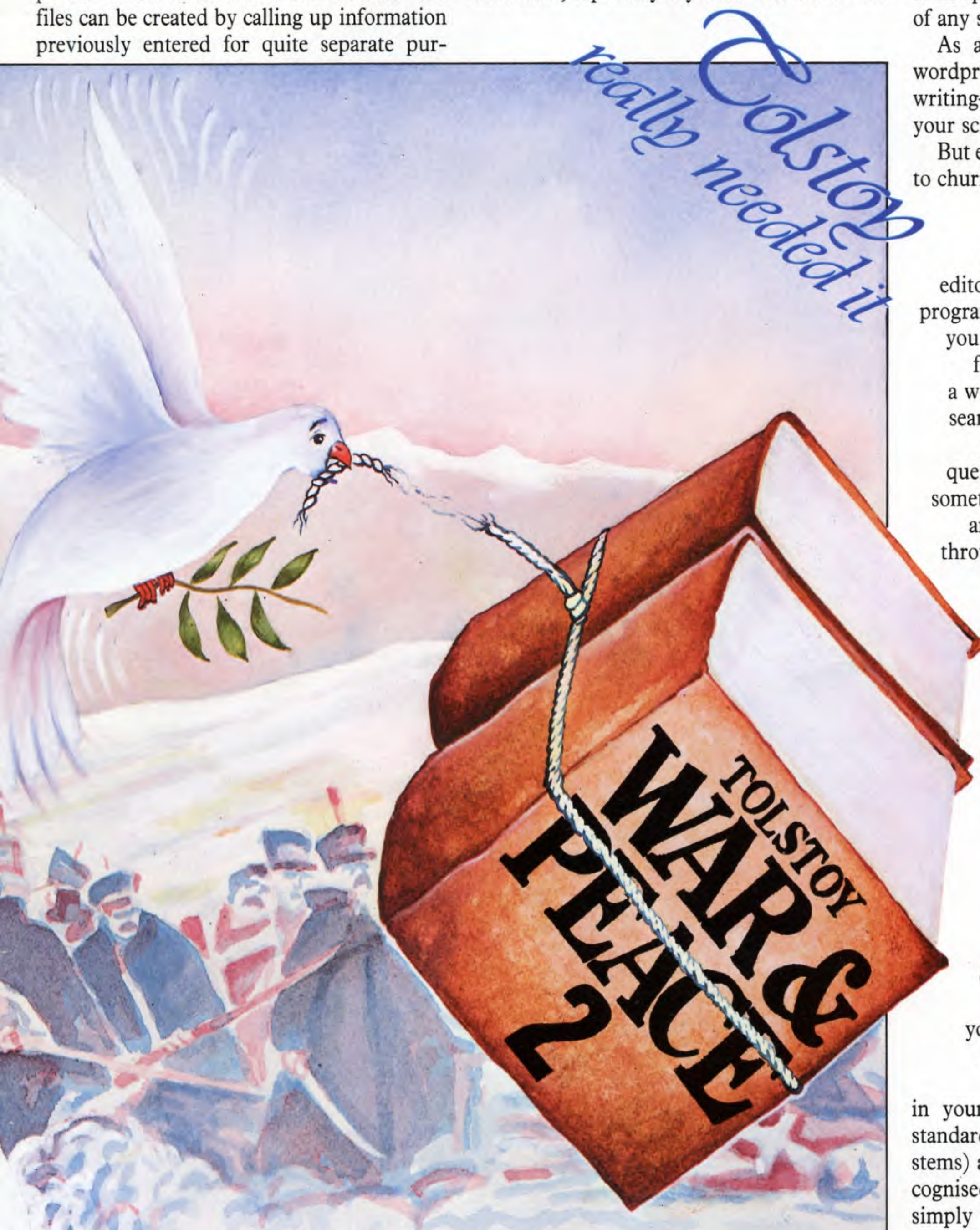


time the program gets to a block, it calls in the fill copy from disk or (more laboriously) from tape to complete the letter. When one letter is completed, the next set of fill copy is called up and automatically inserted into place. So it continues until the batch is finished. Obviously this is very much quicker than either overwriting a stockletter or doing the job manually.

Each person then gets a personalised letter and the same form letter may be used over and over again with fresh or existing fill files.

Interestingly—and this indicates the real power of form letters used for business—fill files can be created by calling up information previously entered for quite separate purposes in a fully-fledged database. An accounts database could thus be used to generate invoices and statements could be sent to account holders. Many database programs can be linked to wordprocessing programs for just this purpose.

At slightly less esoteric levels, databases are used in conjunction with wordprocessing programs to produce things such as a mailing list from which labels can be printed. To fill out several hundred of these—as you might have to do as a club secretary for instance—is, by hand, not one of that office's more pleasurable tasks, especially if you have to do it with any frequency. Once the information has been entered in the mailing list database, it can be called up with ease through a matching word processor.



IS WP FOR YOU?

Now you've got some idea of the potential, is wordprocessing for you? You'll obviously find wordprocessing facilities useful if you're any kind of writer or journalist, or if you run a business, especially if it involves typing many similar documents—acknowledgement letters for orders, contacts, and so on. Much the same applies if you're a club or society official of any sort.

As a student you would probably find wordprocessing helpful for essay or report writing—but get approval and advice from your school or college beforehand.

But even if you're not and don't often have to churn out screeds of text and all you do is type in listings from a magazine, a wordprocessor could have many advantages over the normal program editor. A feature of many wordprocessing programs is a *search* function which enables you to hunt through your copy or listing for a specific sequence of characters—a word, phrase, or variable. Allied to the search function is a *replace* option which you can use to change a character sequence (be it word or a key symbol) into something else. A typical use of the search and replace function would be to hunt through copy looking for, say, an asterisk and replacing this with a frequently used but complicated character sequence, such as may be used in elaborate control commands.

SPECIAL SKILLS?

You don't have to be a good typist, even though wordprocessing will enable you to work very fast if you are. If anything, it's the two-fingered typists who will see the greatest improvement in their typing speed—you can type away, not worrying straight away about the mistakes, which can be corrected at any time before printing. And if you're really bad at spelling, you may even be able to get a compatible spell-check program to run over your copy. This looks at each word in your text file and compares it with a standard dictionary disk and (in some systems) a user-defined dictionary also. Unrecognised words—those either misspelt or simply not resident in the reference dictionaries—are spotlighted for correction.

... AND ORWELL COULDN'T WAIT UNTIL 1984



The value of spell-check programs extends beyond that of simply testing your spelling accuracy. If you have a lot of copy to check through once it's been typed in, a spell-check program can be a much more reliable—and certainly quicker—way of picking up the inevitable errors, especially if the text has been typed in quickly. Also, if you've been working 'on top' of a document for any length of time it is difficult to spot mistakes in areas presumed to be correct.

Many home computers can be used in wordprocessing applications for the price of a program, printer and storage device. A good quality keyboard is essential. Without an accessory keyboard the Spectrum, although otherwise a capable machine, cannot be used for wordprocessing at the same levels as the other machines. Add-on keyboards are available from several makers, however (see page 332). Although you can get similar keyboards for the ZX81, it is really too limited in other respects to use as a wordprocessor.

Then consider the suitability of the computer itself. Quite a bit of memory is required to store longish articles—and of course you need room for the wordprocessing program itself.

With a good quality wordprocessing program occupying, on home computers, up to about 3ØK, not much free memory space remains for your work.

Tape devices, which store program or data files sequentially, can be used but are painfully slow and quite unsuitable in many of the more exotic wordprocessing applications. These may require that data is read in from files which were not necessarily stored in sequential order.

By far the best storage method is a disk unit of one kind or another. The various tape loop systems, such as the Spectrum's Microdrive, come close to providing an alternative but even then do not match the speed and capacity of typical disk systems.

Another significant advantage of disk storage is that it's relatively easy to move chunks of text around from one document to another. This is next to impossible to do using a tape storage system.

Some wordprocessing programs can make

use of disk units as *virtual memory*, holding temporary work files, contents lists and so forth which are called up as required and appended to whatever work you happen to be doing at the time. This is taken a stage further by some programs which automatically save work as memory files, or if you recall other areas of work already in memory.

Virtual memory and linked files (which may be chained together in any order) mean that text size is theoretically limited only by disk capacity. But documents formed in this way and of this size can be unwieldy and slow to work with unless the very programs that permit their use also provide editing facilities which can perform 'global' functions. A global function is one that can be performed not only on the one in memory at present, but also—as an option—on those linked files which follow.

PRINTERS

The printer is an important component of a wordprocessing system, for the type and quality of this have a significant bearing on the standard of presentation possible. For most general purpose 'draft' work, a dot matrix printer is perfectly adequate—some of the better ones are even capable of producing 'correspondence quality' printing. But for the very best quality, a daisywheel type printer can be considered essential. Each has particular advantages and disadvantages, so decide carefully (see pages 225 to 229). Make sure the combined features offered by your wordprocessing program and printer meet your requirements. In scientific and maths work, for instance, the ability to define—and print—subscripts (EG_2) or superscripts (EG^2) is pretty useful.

Before buying a printer or wordprocessing program, check that the two are compatible and that, if a special interface is necessary, this too is compatible.

DISPLAY AND USE

The display on home computers is normally restricted to 32 or 40 characters wide but can be doubled, sometimes, by either hardware or software—perhaps even from within the word processing program itself if a special video display mode can be invoked.

The problem, which is avoided by true word processors which have displays of 80 characters, is that what you type in on screen doesn't directly relate to its true position 'on paper' if you have to work to the narrower constraints of a 32 or 40 character screen.

Accepting that final tidying up can be or is done by the relevant print *formatting* commands, wordprocessing programs for home computers usually adopt one of three different types of display entry.

With all methods, you simply carry on typing as if you were on a very long line, pressing RETURN usually only at the end of a paragraph.

In some systems, characters completely fill the width of the screen, with parts of words spilling over to the next line if they don't fit. This kind of display can be difficult to read off the screen but has the advantage that only the cursor moves during actual entry.

A popular alternative, where the display remains fixed at the normal screen character width, is a technique called *word wrap*. As you reach the last character space of the display during typing, the computer automatically takes the word forward to the next line unless a space (to signify a word end) is reached. The resulting display is of *ragged* copy, ranged left to be flush on the left margin. Spaces at the end of each line are always shorter than the first word of the following line.

Some programs enable you to choose line lengths in excess of the normal display width—a typical maximum is 255 characters—which could be useful for certain 'small print' documents. Not even a really high-resolution display can manage this, and the way round the problem is to use a *window* on the document as a whole. Typically, at any instant the window can look at a 20-line block, 40 characters wide. When you are typing a line and reach the right hand side of the screen, the display appears to shift left a character at a time as a new letter is typed.

Even if you type in your document at, say, 40 or less characters wide, output (the printed copy) can usually be *reformatted* by *control commands* sent to the printer.

This highlights an important point: to a large extent what is shown on a TV screen or monitor doesn't matter, it's the quality of the printout that's important. So, for example, with a computer such as the Dragon, the use of only capital letters (albeit colour coded for different case) need not necessarily be a drawback.

However, for sustained work you want good, clear, upper and lower case letters on the screen. Reading dense lines of text is difficult if the characters are fuzzy, or jitter.

DISPLAY QUALITY

This brings us to another important component of a wordprocessing system: the TV or monitor. Pages 445 to 449 explain the differences between these in detail. One of the advantages, if you can call it that, of using a 40 column (or less) display is that most domestic TVs are capable of handling text at this resolution. Few TVs are capable of handling an 80 column display and a monitor then becomes essential—as it does if you plan to do a lot of wordprocessing. Black and white versions of either a TV or monitor will give a sharper picture, and the simple solution if you use a colour set is to turn down the colour setting. Also, try adjusting the contrast to its highest distortion-free setting.

CONTROL COMMANDS

What about the program itself? Even the simplest type of wordprocessing program, correctly termed a *text editor* has a range of standard features, the commonest of which have already been described.

The text editor becomes rather more of a 'wordprocessor' as the features are piled on—and here what's available seems to be limited only by what the designers have thought could be useful. What these features are, and—very importantly—how these are controlled by the user are the two main criteria when choosing a suitable program.

As well as simply typing in text, directives have to be entered for controlling the features of the program as well as output to the printer. These *control functions* consist of keypresses which are responsible for editing your copy. Good editing commands are crucial to the ease of using a wordprocessing package. Some programs are quite complicated to use and require considerable periods of familiarization—not, therefore suitable for the occasional letter!

Editing commands embrace all the functions of actually writing copy and knocking it into its final form. Printer *control commands* are not, by comparison, 'transparent' to the user and have to be placed within special control markers either as you're writing the copy or afterwards (this is called *post formatting*). These *embedded commands* can look quite strange when displayed and are quite difficult to remember on some programs.

With some wordprocessing programs, the display responds to formatting directives in the same way as a printer would and therefore gives a good idea of what a document looks like when printed. Other programs may simulate this by giving a 'video' preview mode.

What you've got to consider ultimately is features required versus cost. Even cheap wordprocessing programs can perform an amazing number of functions. But more expensive programs can do more, are usually very much quicker in executing these functions and are hence quicker to use once you've got used to them. A later article in this series will cover exactly what's involved in learning to master the art of word processing.

ACORN PROGRAM SQUEEZER

Save memory and speed up your BASIC programs. This machine code routine automatically checks your listings and knocks out redundant REMs or superfluous spaces

When you write BASIC programs, it is very convenient to put in spaces between instructions, variables and data to make the lines easier to read. And REM statements are often included to make things easier to follow—a great help when you're debugging or modifying programs.

The problem is that these spaces and REM statements slow down the programs and take up unnecessary memory space. For maximum efficiency all these unnecessary spaces and lines need to be stripped out. But who wants to go through laboriously pruning them once you've got the program running—possibly introducing new errors?

The following machine code routine is a *stripper*. Once you have got your BASIC program running, you run the stripper and it compresses your program for you.

The following is an assembly language stripper for the BBC Micro and the Electron. As both these computers have assemblers written into their BASIC operating system, the assembly language is entered exactly like BASIC, with line numbers, only the whole of the assembly language routine has to be enclosed in square brackets. It is these that turn the assembler on. Remember to SAVE the program before RUNning it.

HOW IT WORKS

As it is easy to switch back and forth between BASIC and assembly language on the BBC Micro and the Electron, it is very convenient to define variables outside the assembly language program. So the first five lines of this program label five bytes on the zero page. The extensive use of labels instead of memory location addresses and numeric jumps makes the program easier to understand as well as easier to modify and debug.

Line 60 sets up a FOR ... NEXT loop. When you have assembly language programs with forward jumps or branches in them, you must assemble them in two passes—the first time round, the assembler will not be able to fill in



```
10 RFLG=&74
20 QFLG=&75
30 CFLG=&76
40 DFLG=&77
50 COM=&78
60 FOR T=0 TO 3
   STEP 3
70 P%=&900
80 [OPT T
90 STA COM
100 LDA &18
110 STA &71
120 STA &73
130 LDA #0
140 STA &70
150 STA &72
160 LDY #1
170 JMP L3
180 .L2
190 TYA
200 LDY #3
210 STA (&70),Y
220 TAY
230 CLC
240 ADC &70
250 STA &70
260 LDA &71
270 ADC #0
280 STA &71
290 TYA
300 .L25
310 CLC
320 ADC &72
330 STA &72
340 LDA &73
350 ADC #0
360 STA &73
370 LDY #0
380 JSR MVEBYTE
390 .L3
400 LDX #0
410 STX RFLG
420 STX QFLG
430 STX CFLG
440 STX DFLG
450 JSR MVEBYTE
460 CMP #255
470 BNE L35
480 RTS
490 .L35
500 JSR MVEBYTE
510 JSR MVEBYTE
520 .L4
530 LDA (&72),Y
540 CMP #13
550 BEQ L2
560 BIT RFLG
570 BMI BUMP
580 BIT DFLG
590 BMI L5
600 CMP #&F4
610 BEQ RM
620 CMP #34
630 BEQ QUOTE
640 CMP #&DC
650 BNE L43
660 DEC DFLG
670 .L43
680 CMP #32
690 BNE L5
700 LDA COM
710 AND #9
720 BEQ L5
730 AND #1
740 BNE L45
750 BIT CFLG
760 BMI L5
770 .L45
780 BIT QFLG
790 BPL BUMP
800 .L5
810 JSR MVEBYTE
820 CMP #32
830 BEQ L4
840 LDA #255
850 STA CFLG
860 JMP L4
870 .BUMP
880 INX
890 INC &72
900 BNE L4
910 INC &73
920 BNE L4
930 .MVEBYTE
940 LDA (&72),Y
950 STA (&70),Y
960 INY
970 RTS
980 .QUOTE
990 LDA QFLG
1000 EOR #255
1010 STA QFLG
1020 JMP L5
1030 .RM
1040 LDA COM
1050 AND #6
1060 BEQ L5
1070 AND #2
1080 BEQ RM3
1090 BIT CFLG
1100 BPL RM2
1110 DEC RFLG
1120 DEY
1130 JMP BUMP
1140 .RM2
1150 LDY #3
1160 STX RFLG
1170 LDA (&70),Y
1180 SEC
1190 SBC RFLG
1200 JMP L25
1210 .RM3□DEC RFLG
1220 JMP L5
1250 ]: NEXT
```

the values for the labels for forward jumps, so it has to go round again to fill these in.

The two passes are done here with the values Ø and 3 for the variable T. The OPTion command in Line 8Ø tells the assembler whether to report errors and whether to list the machine code. When T is Ø, it neither reports errors nor lists. When T is 1, it does not report errors but does give a listing. When T is 2, it reports errors but does not give a listing. And when T is 3, it both reports errors and gives a listing. T should either be Ø or 1 on the first pass—otherwise it will report the undefined forward jumps as errors—and 2 or 3 on the second pass.

Line 7Ø tells the assembler where to start assembling. P% is the resident integer variable related to the 6502's PC—or program counter—register. And &9ØØ is the address the first byte of machine code is put into.



INITIALIZATION

STA COM STores the contents of the Accumulator into the zero page location defined by COM. When a machine code program is CALLed, the contents of the resident integer variable A% is loaded into the accumulator. So the program can be COMmanded from outside by the value you put in A%.

LDA &18 LoaDs the Accumulator with the contents of memory location 18. And 18 is a system variable which contains the most significant byte of the page in memory where the BASIC program starts. As BASIC always starts at the beginning of a page you don't have to (and can't) look at up the least significant byte. It is always ØØ.

The most significant byte is then stored in &71 and &73 by STA &71 and STA &73. These are the most significant bytes of two pointers the program is going to use. &7Ø and &71 points to where you are in the new, packed program and &72 and &73 points to where you are in the old, unpacked program.

LDA #Ø loads the accumulator with zero. And STA &7Ø and STA &72 sets the least significant bytes of the pointers to ØØ. Both pointers start from the same place.

LDY #1 Loads the Y register with 1. The Y register is going to be used to count along the bytes of the new, packed line. But the first byte of each new line is marked by a memory location containing the byte ØD. So when you start packing a program you can skip the first byte—that's why Y is set to 1 and not Ø.

The program then JuMPs forward to the label L3 in line 39Ø.

LDX #Ø LoaDs the X register with Ø and clears the other four variables by storing Ø in them. Then you Jump to the SubRoutine labelled MVEBYTE.

JSR is the assembly language equivalent of GOSUB, while JMP is the equivalent of GOTO. JSR needs to be followed at some time by a ReTurn (from Subroutine) or RTS instruction. This sends the microprocessor back to the instruction after the JSR.

THE MVEBYTE SUBROUTINE

The subroutine MVEBYTE—in lines 93Ø to 97Ø—not surprisingly moves a byte. LDA

(&72),Y is a complicated and slow instruction. It loads the contents of memory location &72 into the accumulator and adds the contents of the Y register to it. It then loads the contents of &73 into the accumulator and adds any overflow from the first operation to it. In other words, it takes the address store in memory locations & 72 and & 73 and adds the value of Y to it. (The address is stored as usual, in low-high format, so the low byte is stored in location & 72 and the high byte in & 73.)

The contents of the memory location indicated by this indexed pointer are now loaded into the accumulator. As you can see, lots of things seem to be going in and out of the accumulator at the same time. It is complicated, but the 6502 can cope with that, although it takes five machine cycles, compared with two for LDA #Ø, say.

This complex instruction has taken the byte presently being worked on and loads it into the accumulator. On the first pass, for example, &72 and &73 point to the beginning of the old BASIC program and the Y register counts along 1 to the next byte—that is, the byte after the beginning of line marker which is the high byte of the BASIC line number. Remember, BASIC line numbers are stored high-low, rather than low-high like everything else in the machine's memory.

INY INcrements the Y register, ready to cope with the next byte of the program. And RTS returns to the instruction after the subroutine was called—CMP #255 on Line 46Ø. This CoMPares the contents of the accumulator—that is, the high byte of the BASIC line number—with the number 255. So if the most significant byte of the line number is as high as FF in hex, you have reached the end of the BASIC program.

If the high byte is 255, BNE—Branch if Not Equal—does not branch and the microprocessor goes onto the next instruction. RTS returns to BASIC.

If the end of the program has not been reached, the branch is made and the microprocessor skips over the RTS to the instruction which follows it.

Line 5ØØ then jumps to the MVEBYTE subroutine to move the low byte of the line number. Line 51Ø jumps to the MVEBYTE subroutine to store the old line length for future reference.

WHAT HAS BEEN FOUND?

The LDA (&72),Y loads up the next byte and CMP #13 compares it with 13. 13 is the ASCII code for a carriage return, in other words it marks the end of a line. If the next byte is not a carriage return, the BEQ—Branch if EQual—condition in Line 55Ø is not fulfilled and the microprocessor moves onto the next line.

BIT RFLG looks at one of the zero page addresses set aside at the beginning of the program. RFLG stands for Rem FLaG—it is often useful to make your labels meaningful. It makes it easier to understand just exactly what's going on.

This location is being used as a flag in exactly the same way as the bits of the processor register are used as flags. But this time all eight bits are set, under certain





conditions, as a marker. RFLG is set—or filled with 255—when a REM statement is hit. Otherwise it is reset or cleared to Ø by Line 41Ø each time the microprocessor goes round that loop.

BIT puts the most significant bit of the contents in the sign flag of the process register—bit 7 carries the sign. When it is 1, the number is taken as negative in 2's complement, remember. The zero and overflow flags are also affected, but that need not concern you here.

BMI then looks at the sign flag and Branches if MInus. So if RFLG has been set the microprocessor goes and performs the BUMP routine, starting on Line 87Ø.

THE BUMP ROUTINE

This bumps up the beginning of the old line pointer to move it past the current instruction—in this case a REM—onto the next byte. INC &72 increments the low byte and BNE L4 jumps back to the beginning of the routine on Line 52Ø if the result is not zero. If the result is zero—that is, the contents of &72 have been incremented from FF to ØØ—the high byte of the pointer has to be incremented. INC &73 does this and the BNE L4 on Line 92Ø takes the microprocessor back to Line 52Ø again.



The high byte of the pointer can never be zero, so the BNE on Line 92Ø is always fulfilled. But a BNE is used rather than a JMP because it saves a byte.

As you can see, this loop is performed over and over again. Once RFLG is set there is nothing in this loop that can reset it. So the pointer quickly counts down the line. But this is what you would expect. Everything following a REM in a line of BASIC is taken as a comment and can be removed. The only way out of this loop, you'll note, is the test for a carriage return in Lines 54Ø and 55Ø. And each time it goes round the loop, the INX in Line 88Ø clocks up how many bytes are saved—in each individual line.

Lines 58Ø and 59Ø check the Data FLaG in the same way. If DFLG is set the processor branches to Line 8ØØ.

Line 6ØØ contains the instruction that actually checks for a REM in the BASIC program. F4 is the token for REM and CMP #&F4 compares the contents of the accumulator which is still the next byte picked up in Line 53Ø—with F4. And if the next byte is a REM, BEQ RM sends the processor off to the RM routine which starts with the label .RM on Line 1Ø3Ø.

RESPONDING TO A REM

Line 1Ø4Ø then loads the accumulator with the contents of COM. This is where the resident integer variable A% was stored, remember. The value of A% gives you various stripper options. The following table outlines them:

A%	All spaces	REMs + comments	Comments only	Leading spaces only
1	×			
2		×		
3	×	×		
4			×	
5	×		×	
6		×	×	
7	×	×	×	
8				×
9	×			×
1Ø		×		×
11	×	×		×
12			×	×
13	×		×	×
14		×	×	×
15	×	×	×	×

Q+A

Why doesn't the Acorn assembler get mixed up between labels, opcodes and data?

The Acorn assembler divides each assembly language into four fields. These are the **label field**, the **operator field**, the **operand field** and the **comment field**.

The label field does not have to be filled. When it is, it contains the label the processor has to jump or branch to. And the label must have a fullstop in front of it to identify it.

The operator field contains the actual instruction—LDA, STA, JSR or CLC. These operators may or may not need an operand. There must be an operator, if nothing else.

The operand field contains data, an address, a label or nothing at all. The data can be a number like #Ø. The address can be a simple two-byte memory location, the one-byte address of a zero-page memory location or an indexed address with an offset. The label can then mark a position the processor must jump or branch to, or the label that has been given to a particular memory location like RFLG. The operator tells the assembler what sort of operand to expect. An operator like CLC does not need an operator. The comment field is optional and acts like a REM statement. You can write anything you like after the instruction, so long as it will fit a line of BASIC. Instruction and comment must be less than 255 characters.

You will notice that the options are specified by the bit pattern in the least significant nybble of A%.

Line 1Ø5Ø ANDs the command you've fed into A% at the beginning of the program—in other words, it checks bits 1 and 2 to check whether you want either the REM and following comment or just the following comment stripped. This option is given because you might have directed a GOTO or a GOSUB to a REM line. Doing this is bad practice, and you should avoid doing so as it can slow the program down. Lines in loops like this are often performed over and over again and a REM line will waste time on each pass. Still, the option is there if you want to use it.

AND works much the same as AND in BASIC (see page 285). And AND #6 will only return Ø if neither bit 1 or bit 2 is set or you don't want REMs or their comments removed. BEQ is fulfilled on Ø and the processor jumps to Line 8ØØ. And on Line 81Ø the MVEBYTE routine is called. RFLG is not set, so the program will continue to shift bytes as normal.

If that condition is not fulfilled, the program needs to check whether you want REMs *and* comments removed or just the comments alone. To do that the program ANDs the command with 2. If bit 1 is not set, Ø is returned and the BEQ takes the processor off to Line 121Ø. Line 121Ø DECrements the RFLG from ØØ to FF—in other words, sets it—and Line 122Ø then jumps back to Line 8ØØ. The MVEBYTE routine is then executed, which moves the byte containing the REM. But as RFLG is now set, the comment following with REM will be skipped over when the processor hits Lines 56Ø and 57Ø again the next time round the loop.

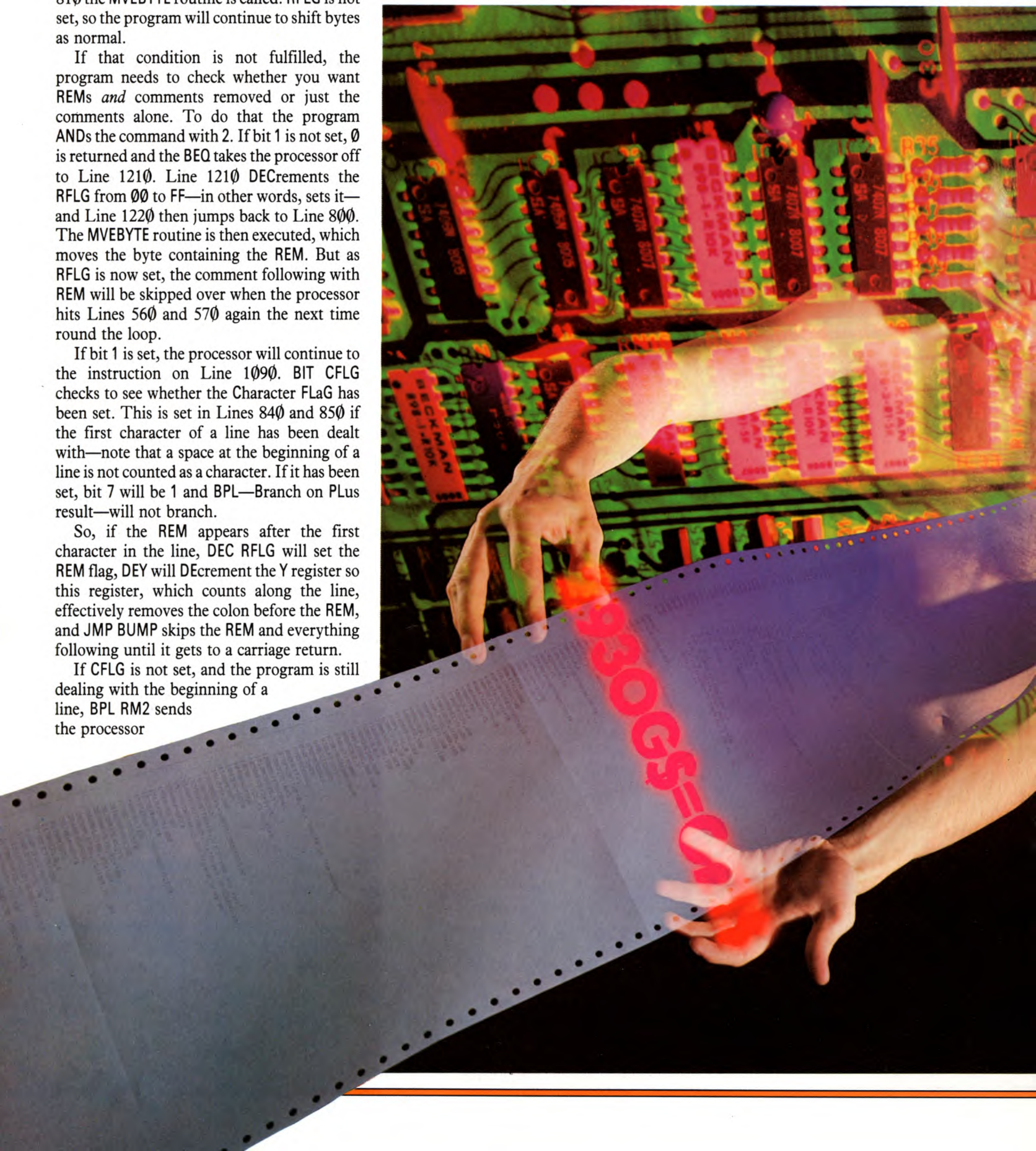
If bit 1 is set, the processor will continue to the instruction on Line 1Ø9Ø. BIT CFLG checks to see whether the Character FLaG has been set. This is set in Lines 84Ø and 85Ø if the first character of a line has been dealt with—note that a space at the beginning of a line is not counted as a character. If it has been set, bit 7 will be 1 and BPL—Branch on PLus result—will not branch.

So, if the REM appears after the first character in the line, DEC RFLG will set the REM flag, DEY will DEcrement the Y register so this register, which counts along the line, effectively removes the colon before the REM, and JMP BUMP skips the REM and everything following until it gets to a carriage return.

If CFLG is not set, and the program is still dealing with the beginning of a line, BPL RM2 sends the processor to the label .RM2 on Line 114Ø.

LDY#3 then loads the Y register with 3, to point at byte three in the line. STX RFLG stores the contents of the X register in RFLG. The contents of byte three of the new line are then put into the accumulator. This was the old line length, remember, which was transferred directly as it was going to be filled in later.

SEC then SEts the Carry flag. This should always be done before a subtraction is carried out because it is the 1 from the carry flag that gives the 1 that's added to the flipped bits to



give 2's complement (see page 181). When the processor subtracts, it flips the bits and adds. And any addition takes the carry flag into account. So if the carry flag is set to 1, the 1 is added—effectively giving 2's complement. And when the carry flag is not set—that is,



it's Ø—then Ø is added in. That effectively subtracts an extra one.

This is very useful if you are subtracting two-byte numbers. If you set the carry, subtract the low bytes first, then the high bytes. Then if the low bytes need to borrow the carry flag will automatically be set to Ø. When the high bytes are subtracted, an extra 1 will be taken away and the borrow will automatically be accounted for this way. You will note that the carry flag works the other way round from what you would expect logically. When it is set to 1, there is no borrow, and when it is reset to Ø, there is.

So the rule is: If you're going to subtract, set the carry flag. And if you're going to add, clear the carry flag—that is, reset it to Ø. You don't want an extra 1 added in then, after all. On the 6502, there are no adds or subtracts without carry.

SBC RFLG then SUBtracts the number of bytes you've bumped along the line stripping the REM statement or spaces. INX clocked that up in Line 88Ø, remember, and the instruction in Line 116Ø put X in RFLG. The processor then jumps to Line 3ØØ.

SETTING THE OLD LINE POINTER

CLC CLears the Carry flag—the next instruction is an add. ADC &72 ADds with Carry the contents of memory location 72 hex and the accumulator, and leaves the result in the accumulator. That is, it adds the length of the line to the low byte of the pointer that is keeping track of where you are in the old program. STA &72 stores the result back in the pointer. LDA &73, ADC #Ø and STA &73 simply updates the high byte of the pointer if there is an overflow from the low byte. ADC #Ø adds Ø to the high byte when it's in the accumulator if there is no carry, but it adds 1 if there is. The carry can only have come from a low-byte overflow.

JSR MVEBYTE in Line 38Ø gets the ØD line marker in the first byte of the next line moved into the new, packed program. Then you are ready to start all over again on the next line.

If there is no REM, the program does not branch in Line 61Ø. The next thing it checks for is quotes. Whatever is contained in quotes is a string and you don't want to take spaces out of strings, otherwise it would affect things you PRINTed on the screen.

THE QUOTE ROUTINE

The contents of the accumulator—which are still the next byte of the old program line picked up in Line 51Ø—are compared with 34 by the instructions in Lines 62Ø and 63Ø. 34 is the ASCII for quote marks and the BEQ instruction in Line 63Ø branches to the QUOTE routine which starts on Line 98Ø if the next character is quotes.

This loads the contents of the zero page memory location QFLG—or Quote FLaG—into the accumulator. QFLG works in the same way as RFLG. It is either Ø or 255. This time though, it is set to 255 on odd numbers of quotes and reset to Ø for even numbers. What follows an odd number of quotes is in quotes. But what follows an even number of quotes is not a string, it's part of the program so the spaces can be stripped.

EOR is an Exclusive OR. This works exactly the same way as EOR in BASIC (see page 287). In other words, if QFLG contains 255, EOR #255 gives Ø. And if QFLG contains Ø, EOR #255 gives 255. STA QFLG stores the result back in QFLG.

So if the QFLG is set—that is, the program has hit an odd number of quote marks in the line already—when it hits another it now has passed an even number, so it clears the quote flag. But if the QFLG is not set—and an even number of quotes has been encountered so far—when it meets another one, there is now an odd number and the QFLG is set.

After that the processor always jumps back to L5 in Line 8ØØ.

Line 81Ø gets the next byte moved. That done, you check to see whether the last byte moved was a space—32 is ASCII for a space. If it was, the BEQ L4 in Line 83Ø branches back to L4 in Line 52Ø and starts all over again.

LDA #255 and STA CFLG sets the Character FLaG, so that it knows when it has passed the first character in the line. This is done because you have an option which only strips the leading spaces and you've got to be able to tell the difference.

If the last character moved was not a space, the instructions in Lines 84Ø and 85Ø set CFLG—the Character FLaG. The JMP L4 takes it back to the beginning of the routine again.

DATA STATEMENTS

When the character encountered is not a quote mark, the branch is not made in Line 63Ø. The next thing that is checked for is DC—the token for DATA. Strings can be stored in DATA statements and again you don't want to strip spaces from strings.

If the next byte is not a DATA, the BNE branches over to Line 67Ø. Otherwise, the DEC DFLG in Line 66Ø sets the Data FLaG. Either way, the instruction in Line 68Ø checks for a space again, and the BNE L5 branches to the usual byte moving routine if the next byte is not a space.

If it is, AND #9 checks the command set on the zero page. Zero is only returned if

Microtip

Take care to specify the origin correctly

When you use the Acorn's built-in assembler, it is important that you specify the origin of the machine code routine within the assemblers double-pass FOR . . . NEXT loop. That is

P% = start address

must follow

FOR T = Ø TO 3 STEP 3

Otherwise, when the assembler makes its second pass it will assemble machine code program again, directly after the first-pass's hex. This leaves the two programs end to end.

More than likely, doing this will kick up an error message. When the assembler fills in the labels on the second pass it will put in the relative jump to destinations in the first program and these are likely to be greater than 128 bytes.

neither bit Ø—strip all spaces—nor bit 3—strip leading spaces only—is specified. In that case, the BEQ L5 in Line 72Ø branches to the usual byte moving routine starting in Line 8ØØ.

And if that does work, AND #1 checks if the command is 'strip leading spaces only'. If it is the BNE L45 in Line 74Ø branches to the routine starting on Line 77Ø. That checks whether the quote flag has been set. And if it hasn't, it branches to the BUMP routine which skips over the space. This effectively removes it from the new, packed version of the program.

If the command byte does specify that leading spaces should be stripped, the program, not unnaturally, checks to see if it is at the beginning of a line. And how it does that is check whether the character flag, CFLG, is set. If it is—and the program has passed the leading spaces and is into the line proper—the BMI instruction, Branch on MInus result, on Line 76Ø, sends the processor onto the usual byte moving routine. Otherwise BIT QFLG checks whether it is in a string or not, and if it's not, BPL BUMP skips over the space and does not copy it into the new, stripped version of the program. Otherwise, the program moves onto the usual byte moving routine.

So the routine labelled L5 gets the bytes moved in the normal course of events by calling the MVEBYTE routine. And the routine labelled L4 examines the various options each time around the loop. They call each other, and between them they perform the strip.

At the end of each line, though, the instruction in Line 54Ø hits the byte 13. This is the carriage return that marks the end of a line of BASIC and the instruction in Line 55Ø calls the routine L2 which starts on Line 18Ø. This Transfers the contents of the Y register into the Accumulator with the instruction TYA, then loads Y with the number 3. The contents of the accumulator are then stored in the memory location given by &7Ø offset by 3.

CLEANING UP THE LINE

&7Ø is the pointer that points to the beginning of the latest, new, packed line of BASIC. Its third byte should contain the line length. But at the moment it contains the old line length which was transferred direct from the old line. The Y register has been counting up the length of the new line though. These three instructions now slot its value into the third byte, giving the correct new line length.

The line length value is then stored back in the Y register with TAY, so that you can work with the accumulator without losing it. Lines 23Ø to 28Ø then increment the new line pointer in zero-page memory locations &7Ø and &71—in exactly the same way as &72 and &73 in Lines 31Ø to 36Ø were incremented as mentioned above—so that they point to the beginning of the next line of BASIC, which starts at the address given by the start of the last line plus the line length.

This addition is performed in the accumulator. So to get the value of the new line length back into the accumulator, TYA transfers it back from the Y register again. Lines 31Ø to 36Ø then increment the pointer in &72 and &73, which now contains the address of the beginning of the old line of BASIC plus any bumping that has been done. Anyway, after it has been incremented by the new line length, it points to the beginning of the next line of BASIC. The program is now ready to start all over again stripping the next line.

This whole process continues until the instructions in Lines 45Ø and 46Ø hit a 255, which marks the end of the BASIC program. The BNE L35 following it then does not branch and the RTS returns to BASIC with the program stripped.

HOW TO USE IT

To assemble the stripper, RUN this program in the normal way (SAVE it first for future additions next time). As it RUNs it will list the assembly language with the hex alongside.

When it has finished, you can NEW it to get rid of the BASIC and assembly language given above. Then feed in the BASIC program that you want to strip. Specify whether you want all the REMs removed, or just the comments following. Also specify whether you want all the spaces removed, or just the leading ones, by setting A% with one of the values shown in the table above. For example:

```
LET A%=15
```

will get rid of all REMs and all spaces. Then CALL the stripper with:

```
CALL &9ØØ
```

You must do this only after you are satisfied that the BASIC program is working properly. You won't simply be able to edit some of the lines after the program has been stripped.

As with all the programs LISTed in *INPUT* you should SAVE it. To SAVE machine code you need to use the command:

```
*SAVE "STRIPPER" 9ØØ+cc
```

where cc is the length + 1 of the machine code program you're SAVEing. You'll be able to work this out when the program is assembled. The assembler listing will tell you the address of the last instruction of the program. If this is a one byte instruction—like RTS or CLC—the address given for it is the end address of the program. But if it's more than one byte long—and you'll see how many bytes from the listing too, each pair of hex digits the instruction is translated into is one byte—you'll have to add on the extra to get the end address. And to get the length of the program you have to subtract the end address from the start address—&9ØØ—and add one. It doesn't matter if you save an extra few bytes after the end of the program though. So if you're worried about your arithmetic it's better to err on the generous side.

The code is then LOADed back from tape by the command:

```
*LOAD "STRIPPER"
```

In this case, though, you should SAVE the assembly language listing—or source code—as well. In the next issue of *INPUT* another routine will be added to this stripper which will tell you how many bytes you have saved by removing the REMs and spaces. To do this, some of the lines given above will have to be overwritten.

The assembly language program (and its assembly instructions) given above is essentially the same as BASIC, so you can SAVE it to tape in the normal way.

LET PLAY COMMENCE

- DEALING CARDS
- PLACING A BET
- BUY, TWIST AND STICK
- BREAKING THE BANK
- RUNNING TOTALS

The banker gives you his glassy-eyed stare. You make your bet and receive another card, but do you buy, twist, or stick? This time you'll see the lines for the player

Following on from the graphics routine you typed in last time, you'll need two more sections of program—one to handle the player's responses, and one to enable the computer to play. In this article you will be shown the lines you'll need for the player. But you won't get very far with playing the game at this stage because you haven't taught the computer how to play yet.

This section of program has to do a number of different jobs. (Don't worry if you aren't sure of the exact rules of Pontoon—these will be covered together with the final section of programming in the last part of this article.) Basically, the program has to do three things. It has to handle the deal, allow the player to place a bet and ask for additional cards—finally it has to work out the player's score.

The section of program which follows asks the player to place a bet on the first card after the computer (the banker) has dealt one to the player and to itself. After receiving the second card, the player is offered the options of sticking, twisting, or buying, according to the state of play.

The program has routines for handling each of the options, dealing extra cards and/or adding to the stake, or passing on to the banker's turn. After each extra card, the program checks in case the score has exceeded 21 and the player has bust.

In addition, the program checks for special conditions—pontoons and five-card tricks. If either of these occur, a message is displayed to tell the player what has been scored, and the turn passes on to the banker.

Now add the lines overleaf to the program—Line 530 has been extended from last time:



```
90 DIM S(2): LET BET=B
100 DIM O(2): DIM W(2)
500 LET Y=B: LET X=C: LET TF=B: LET
    AF=B: LET PF=B: LET FF=B
520 FOR U=C TO 2
530 GOSUB 5500: GOSUB 6000: GOSUB
    6500
535 IF CC=52 THEN LET CC=B
540 LET O(U)=C(CC+C)
550 LET X=X+6
560 PRINT PAPER 7;AT 21,B;"YOU
    HAVE□";CP;"□CHIPS□"
570 IF U=C THEN INPUT "WHAT IS YOUR
    BET?□";AM: IF AM>CP OR AM<C
    THEN GOTO 570
590 IF U=C THEN LET CP=CP-AM:LET
    BET=AM
595 GOSUB 7000: GOSUB 7000: NEXT U
622 IF S(2)=21 THEN LET PF=C: PRINT
    PAPER 2;AT 4,18;"□PONTOON□":
    GOTO 2500
700 IF (S(C)<16 AND (S(2)<16 OR
    S(2)>21) OR TF=C OR CP<AM) THEN
    GOTO 705
702 INPUT "BUY, TWIST OR STICK?□";
    LINE D$:IF D$<>"B" AND D$<>"T"
    AND D$<>"S" THEN GOTO 702
703 GOTO 715
705 IF (S(C)>21 OR CP<AM OR TF=C)
    THEN GOTO 710
706 INPUT "BUY OR TWIST?□"; LINE D$: IF
    D$<>"B" AND D$<>"T" THEN
    GOTO 706
708 GOTO 720
710 IF (S(C)<16 AND (S(2)<16 OR
    S(2)>21)) THEN GOTO 770
711 INPUT "TWIST OR STICK?□"; LINE D$:
    IF D$<>"T" AND D$<>"S" THEN
    GOTO 711
715 IF D$="S" THEN GOTO 2500
720 IF LEN D$=B THEN GOTO 700
725 IF D$(C)<>"B" THEN LET TF=C:
    GOTO 905
760 LET BET=BET+AM: LET CP=CP-AM
770 PRINT PAPER 7;AT 21,B;"YOU
    HAVE□";CP;"□CHIPS"
905 LET Z=C(CC)
910 GOSUB 5500: GOSUB 6000: GOSUB
    7000
920 LET X=X+6
921 IF S(C)>21 THEN GOTO 2000
925 IF X=31 AND S(C)<22 THEN PRINT
    PAPER 2;AT 21,B;"□FIVE-CARD
    TRICK!□": LET FF=C: GOTO 2500
930 GOTO 700
2000 IF CP=B THEN PRINT AT
    21,B;"YOU'VE LOST ALL YOUR CHIPS!":
    STOP
2010 IF CP>=1000 THEN PRINT AT
    21,B;"WELL DONE! YOU BUST THE
    BANK": STOP
2020 PRINT #C;"PRESS 'S' FOR ANOTHER
    HAND"
2030 IF INKEY$<>"S" THEN GOTO 2030
2037 IF PF=C OR DPF=C THEN GOSUB
    5000
2040 CLS : GOTO 90
2500 STOP
6000 IF VA>10 THEN LET VA=10
6010 LET S(C)=S(C)+VA: LET
    S(2)=S(2)+VA
6020 IF VA=C AND AF=B THEN LET
    S(2)=S(2)+10: LET AF=C
6030 IF S(C)>21 THEN PRINT FLASH C;
    PAPER 7;AT 20,B;"□";TAB 9;"YOU'RE
    BUST!□";TAB 31;"□": LET DPF=B:
    RETURN
6040 PRINT AT 20,B;"□";TAB 31;"□"
6050 PRINT PAPER 7; INK B;AT 20,B;"YOUR
    SCORE IS□";S(C);: IF S(2)<>S(C)
    AND S(2)<22 THEN PRINT PAPER 7; INK
    B;"□OR□";S(2)
6060 RETURN
6500 FOR N=10 TO 18
6510 PRINT PAPER 7; INK 1;AT N,X;CHR$ 161;
    CHR$ 161;CHR$ 161;CHR$ 161;CHR$ 161
6520 NEXT N
6530 RETURN
7000 LET CC=CC+C: IF CC=53 THEN LET
    CC=C
7010 RETURN
```

Lines 90 and 100 set up the BET variable and the three arrays—S contains two score totals for the player (an ace can count for either one or eleven), O contains the computer's two initial cards, and W contains the two scores from those two cards (if an ace is present).

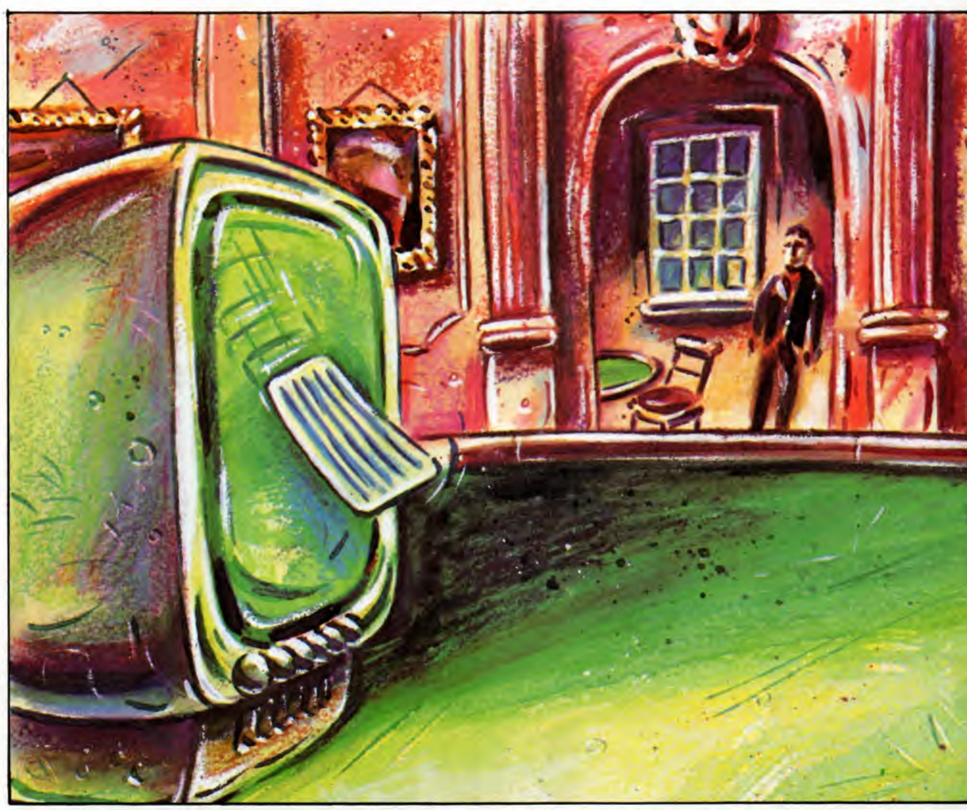
MORE CARDS

Line 500 sets up the coordinates for the top left-hand corner of the card, and sets four flags to zero. TF is the 'twist' flag and is set to one as soon as the player elects to twist during the game (the player will now be prevented from buying cards), AF is the ace flag (used so that two totals can be calculated, with ace being either one or 11), PF is the pontoon flag, indicating if the player has pontoon, and FF is the five-card trick flag.

There is a FOR ... NEXT loop between Lines 520 and 595 which looks after dealing the first two cards to the player and banker. Line 530 now calls two extra subroutines: GOSUB 6000 updates the score as the cards are dealt, and GOSUB 6500 displays the reverse side of the cards—the banker's cards.

In detail, Line 6000 checks if the card is a picture card, and then gives it the value ten. Line 6010 adds the value to the two elements of the score array. Aces are checked for by Line 6020. Line 6030 checks if the player is holding more than 21. If he is, then he is told YOU'RE BUST.

After RETURNing to the main body of the



program, Line 535 checks if the current card is number 52, and changes it to zero if it is. Line 54Ø puts the computer's cards into array O. Line 55Ø moves the next card six screen positions to the right.

BETTING

The number of chips the player is holding is displayed by Line 56Ø, whilst Line 57Ø asks for the bet, but only after the first card. The line also makes sure that the player is holding sufficient chips for the bet. The bet is subtracted from the chip total, and the BET variable itself is set up in Line 59Ø. The two variables AM and BET are needed to keep track of how the player is buying cards, if that option is chosen later.

The two GOSUB 7ØØØs in Line 595 adjust CC, the current card, and make sure that CC returns to the beginning of the pack if it becomes too large. The FOR ... NEXT loop ends here.

Line 622 checks if the two cards that have been dealt to the player make a pontoon. The pontoon flag, PF, is set, and then the program stops—the section from Line 25ØØ onwards is added in the next article.

Lines 7ØØ, 7Ø5 and 71Ø check various combinations of score, twist flag and available chips. Their role is to point to the correct combination of options to be presented to the player. For example, if the scores are over 16 *and* below 21, *and* the player hasn't yet twisted, *and* has sufficient chips to buy a card, Line 7Ø2 gives the player the option of buying, twisting or sticking. If, on the other hand, the player has already twisted a card, the buy option is no longer open, so Line 711 gives the player the option of twisting or sticking. The final combination of options is twisting or buying if neither score in S has exceeded 16. These options are presented by Line 7Ø6. If the player elects to stick at this stage, this cut-down version of the program ends.

Line 72Ø checks if the player has just hit [ENTER] instead of any other key. The program goes back to offering the options. Line 725 sets the twist flag if the player hasn't bought a card. The program then jumps to Line 9Ø5.

If the player has chosen to buy a card, the program arrives at Line 76Ø, adjusts the BET, and the number of chips—CP. Line 77Ø displays the number of chips remaining.

Lines 9Ø5 to 92Ø display the extra cards—the ones that have been twisted or bought—on the screen. Line 921 adds up the score. If it's over 21, the program jumps to Line 2ØØØ—the start of the checks after the player has bust.

Line 925 is the check for a five-card trick. This is done by checking the screen position of the last card dealt. If this is in the fifth card position, and the score is below 22, the player has a five-card trick. The five-card trick flag is set and the program jumps to Line 25ØØ. If the player hasn't elected to stick, and hasn't accumulated a five-card trick, Line 93Ø sends the program back to Line 7ØØ.

BREAKING THE BANK

Line 2ØØØ checks if the player has lost all his supply of chips, and says so if he has. The game ends when the player has no more chips. Line 2Ø1Ø checks for a happier outcome: if the player has more than 1,000 chips the bank has been broken, and the game ends too.

If the player still has some chips left, but hasn't accumulated more than 1,000, Line 2Ø2Ø asks if another hand is wanted. PRINT #1 will put the message on the first of the two normally unusable lines at the bottom of the screen.

If either the player or the dealer has managed to score pontoon, the pack is shuffled by Line 2Ø37 calling the shuffling subroutine. If the player wants another go, Line 2Ø4Ø clears the screen and starts again.

Add these lines to the program you have already entered. The player will be able to play one half of the game, but the computer won't be able to reply until next time.

```
140 Z=Ø:GOTO 760
220 T=T+JM
230 FOR C=Ø TO 159:POKE 1Ø24+C,
    32:NEXT C
31Ø POKE 198,Ø
32Ø IF JJ>9 THEN TU=TU+1Ø
33Ø IF JJ<1Ø THEN TU=TU+JJ:IF JJ=1
    THEN T2=1
34Ø PRINT "■■";S2$;"□TOTAL:";
    TU;:T3=Ø:IFT2=1THENT3=1Ø:
    IFTU+T3>21THENT3=Ø
35Ø IF T2>Ø AND TU+1Ø<22 THEN PRINT
    "OR";TU+1Ø
36Ø IF TU+T3=21 AND NU=2 THEN
    PRINT"■■■PONTOON!":
    PO(PL)=1:GOTO 8ØØ
37Ø IF TU>21 THEN PRINT
    "■■■";S1$;"'VE BUST!": GOTO 8ØØ
38Ø IF NU=5 THEN PRINT
    "■■■FIVE-CARD TRICK!":
    CT(PL)=1:GOTO 8ØØ
4ØØ IF TU=21 THEN PRINT
    "■■■YOU HAVE 21!": GOTO 8ØØ
42Ø IF PL=1 AND NU=1 THEN 97Ø
43Ø K1$="T":K2$="T":PRINT
    "■■▒";:IF EB=Ø AND MU>=BE
    THEN PRINT "(B)UY□";:K1$="B"
44Ø PRINT "(T)WIST□";
45Ø IF TU>15 OR TU+T3>15 THEN PRINT
    "(S)TICK";:K2$="S"
46Ø PRINT "?":POKE 198,Ø
```



```
470 GET K$:IF K$< > K1$ AND K$< >"T"
    AND K$< >K2$ THEN 470
480 IF K$="S" THEN PRINT
    "■■■■OKAY.":GOTO 800
490 IF K$="T" THEN EB=1
500 IF K$="B" THEN
    TB=TB+BE:MU=MU-BE
760 TB=0:EB=0:FOR PL=1 TO 2
770 IF PL=1 THEN S1$="YOU":
    S2$="YOUR"
790 TU=0:NU=0:T=-JM:T2=0:
    T3=0:PO(PL)=0:CT(PL)=0:
    GOTO 150
800 RUN
970 CH$="CHIPS":IF MU=1 THEN
    CH$="CHIP"
980 PRINT "■■■";CH$;
    ":■";MU
990 BE=0:INPUT "■■■■ENTER
    BET";BE:IF BE>MU OR BE=0 THEN 990
1000 TB=BE:MU=MU-BE:GOTO 180
```

If you own a Vic 20, you should make sure that Line 230 reads as follows so that the display is adjusted correctly:

```
230 FOR C=0 TO 87:POKE 7680+C,32:
    NEXT C
```

Line 140 sets Z to zero. This makes sure that the program starts dealing from the top of the pack. The program jumps to Line 760 which sets the total bet, TB, to zero, along with the twist flag, EB. In addition, there's the start of a FOR . . . NEXT loop. The program, as it stands, may look a little peculiar because you haven't yet added the NEXT to close the loop. It isn't good programming practice, but remember that the program is still being developed.

Line 770 sets S1$ and S2$ for player one. The total and the card number are set to zero ready for the start of the player's hand. This is set to −JM so that the displays appear correctly. T2 and T3 are set to zero—when an ace is dealt T2 is set to 1, as a flag, and T3 to 10, as a correction for when an ace is chosen to count as 11. PO and CT are also cleared at this stage. The program then goes back to Line 150, when the cards are dealt.

Line 220 calculates the position of the next card, before Line 230 clears the top four lines of any text that has been displayed. The keyboard buffer is cleared by the POKE in Line 310.

Now that a card has been dealt, the program makes a few checks. Firstly, if the card is a ten or a picture card, ten is added to the running total by Line 320. If the card isn't a ten or a picture card, Line 330 adds the card's face value to the running total. If the card is an ace, T2 is set to 1.



RUNNING TOTALS

The total is displayed by Line 340. Ten is added to the running total if an ace is present. If that total is greater than 21, it is ignored, but if it isn't, the second score is displayed by Line 350. Line 360 checks for pontoon and sets the pontoon flag, PO, if it finds one. Similarly, Lines 370 to 400 check for busting, five-card tricks and 21, and PRINT out an appropriate message.

Lines 430 to 460 are concerned with offering the player the options of buying, sticking or twisting. If the player hasn't twisted, and has a total below 21, the options of buying or twisting are offered; if the score is 16, or above, the option of sticking is also offered. If the player has elected to twist at any stage, the option of buying a card is not offered.

Lines 470 to 500 handle the player's reply to the options given. If the player elects to buy a card, Line 500 changes the chip totals.

BETTING

Lines 970 to 1000 handle the bets. Lines 970 and 980 display the number of chips remaining. Line 990 prompts the player to place a bet on the first card, and checks if there is sufficient funding for the bet. Line 1000 sets TB equal to the bet, and subtracts the bet from the number of chips.

These are the extra lines to add to the Acorn program that you have already entered:

```
10 M=100
270 PROCCB(100,600):PROCCB(100,100)
280 TT=0:TT2=0:N=0:E=0:BF=0:
   PP=600:TP=100
290 PROCTWIST
300 PROCBET:PROCCB(300,600):
   PROCCB(300,100):PROCTWIST:
   PROCRESPONSE
310 IF TT=21 AND N=2 THEN
   PRINT'"PONTOON"
320 IF E=1 THEN PRINT"FIVE-CARD TRICK"
340 IF E=2 THEN PRINT"YOU BUST": GOTO
   470
470 REM
480 IF M<1 THEN PRINT'"YOU'RE
   BROKE":END
490 IF M>999 THEN PRINT"YOU HAVE
   BUST THE BANK AND THIS TABLE
   IS□□□□CLOSING":END
500 PRINT"DO YOU WANT ANOTHER HAND
   (Y/N) ?"
510 G$=GET$:IF G$="N" THEN
   PRINT"YOU FINISHED WITH□";M;
   "□CHIPS":END
520 IF G$<>"Y" THEN 510
530 MODE1:PROCSCREEN
550 GOTO 270
930 DEF PROCRESPONSE
940 CLS
950 PRINT"YOU HAVE□";TT;:IF TT2<>0
   THEN PRINT"□OR□";TT2 ELSE PRINT
960 IF E<>0 THEN ENDPROC
970 PRINT"(T)WIST";
980 IF TT>15 THEN PRINT
   "□(S)TICK";
990 IF BF=0 AND M>=B THEN
   PRINT"□(B)UY" ELSE PRINT"□?"
1000 G$=GET$
1010 IF G$="S" AND TT>15 THEN
   E=3:ENDPROC
1020 IF G$="B" AND BF=0 AND
   M>=B THEN PROCCB(TP,600):
   M=M-B:BET=BET+B:
   GOTO 1050
1030 IF G$<>"T" THEN 940
1040 BF=1
1050 PROCTWIST
1060 GOTO 940
1070 DEF PROCTWIST
1080 PROCCARD(TP,PP,CN)
1090 N=N+1:V=V(CN)
1100 CN=CN+1:TP=TP+200
1105 IF CN=53 THEN CN=1
1110 TT=TT+V
1120 IF TT=21 AND N=2 THEN
   TT2=0:E=3:ENDPROC
1130 IF V=11 AND TT2=0 THEN
   TT2=TT-10:GOTO 1160
1140 IF V=11 AND TT2<>0 THEN
   TT=TT-10:TT2=TT2-10
1150 IF TT2<>0 THEN TT2=TT2+V
1160 IF TT<22 THEN 1200
1170 IF TT2=0 THEN E=2:ENDPROC
1180 TT=TT2:TT2=0
1200 IF N=5 THEN E=1
1210 ENDPROC
1320 DEF PROCBET
1330 CLS
1340 PRINT"YOU HAVE□";M;"□CHIPS"
1350 INPUT"HOW MUCH DO YOU WANT TO
   BET□",B
1360 IF B<1 THEN PRINT"YOU HAVE TO
   BET AT LEAST 1":GOTO 1350
1370 IF B>M THEN PRINT"YOU ONLY
   HAVE□";M;"□CHIPS":
   GOTO 1350
1380 BET=B:M=M-B
1390 ENDPROC
1450 DEF PROCCB(X,Y)
1460 GCOL0,2
1470 FOR X2=X TO X+152 STEP 8
1480 MOVEX2,Y:DRAW X2,Y+304
1490 NEXT
1500 GCOL0,3
1510 FOR Y2=Y TO Y+304 STEP 8
1520 MOVEX,Y2:DRAW X+152,Y2
1530 NEXT
1540 ENDPROC
```

Line 270 calls PROCCB twice. PROCCB is situated between Lines 1450 to 1540 and displays a striped card back each time it is called. The coordinates of the bottom left-hand corner of the card are passed to the subroutine through variables X and Y every time the procedure is called. There are now two card backs displayed on the screen.

Line 280 sets up a number of variables and flags. TT and TT2 are totals—the numbers arrived at by adding up the values of the cards being held. TT and TT2 are needed because an ace counts for one or 11, so you may need two running totals; N is the number of cards dealt so far; E is given a range of values which indicate if the player has a five-card trick, pontoon, or has bust; BF is the buy flag which is set to one if the player has twisted—the player will now be prevented from buying cards—and PP and TP are the X and Y coordinates of the card's bottom left corner.

MORE CARDS

Line 290 calls PROCTWIST. PROCTWIST is situated between Lines 1070 and 1210 and *doesn't* just handle the twist option. It's used every time a card has to be turned over on the screen. Line 1080 calls PROCCARD which displays the front face of the card—you've already entered this PROCedure. N is incremented and the value, V, of the card is set by Line 1090 while the card number is incremented and the X coordinate of the card is increased by 200 pixels so that the next card will be displayed at the correct position on the screen by Line 1100. The total is increased by the value of the new card in Line 1110.

Line 1120 checks for a pontoon. It does this by testing if TT is 21 and only two cards have been dealt. TT2 is cleared and E is set to three, indicating that the player has pontoon. Lines 1130 and 1140 adjust TT2 and/or TT according to whether one or two aces have been dealt. Line 1150 increases TT2 if it is already being used, and Line 1160 checks that TT is less than 22.

Line 1170 checks if TT is zero. If TT is zero at this stage, it means that the player has bust. If you trace the program through step-by-step you should see why this is. E is set to 2. If the player hasn't bust, Line 1180 sets TT=TT2 and then clears TT2.

The last part of the subroutine checks the card number. If it's 53, then it's reset to 1. Line 1200 checks for a five-card trick and sets E to 1 if it finds 1.

With the first card, though, much of this PROCedure is redundant, but there is no harm

in using it and it does save having to write one specific routine for turning over the first card. Line 3ØØ calls PROCBET which first tells the player how many chips he has—Line 134Ø. The player must then tell the computer how much he wants to bet. Line 136Ø checks that the bet is at least one chip, and Line 137Ø stops the player betting more chips than he has. Line 138Ø sets up the BET and subtracts the bet from the amount of chips there were at the start.

Following PROCBET, Line 3ØØ deals two more cards using PROCCB, and turns one over using PROCTWIST—this time all the checks *are* needed. Next, the program uses PROCRESPONSE to prompt the player to buy a card, twist or stick—the exact combination of options depends on what the player has done so far, and the value of the cards being held.

RESPONSES

Looking more closely at PROCRESPONSE—starting at Line 93Ø—the player is told the value of the cards being held—two values if an ace is present. Line 96Ø ends the PROCedure if there is a pontoon or a five-card trick, or the player has bust. Line 97Ø displays TWIST as this option will always be open to the player. If the player is holding cards totalling 16 or more, Line 98Ø gives the player the option of sticking. Similarly, if the player hasn't yet twisted—BF=Ø—and has sufficient chips, the option of buying a card is given by Line 99Ø.

Line 1ØØØ prepares the machine for the player's response. Line 1Ø1Ø allows the player to stick if the card total is enough. The value of E is set to three, signifying that the player has stuck. Line 1Ø2Ø lets the player buy a card if no twists have been asked for and there is enough money left. A card is dealt by PROCCB, the number of chips is adjusted, along with the value of BET, and the program jumps to Line 1Ø5Ø which calls PROCTWIST. Line 1Ø3Ø makes sure that the player has responded with a T, S or B—if he hasn't, then the prompts are displayed again by jumping back to Line 94Ø. The program reaches Line 1Ø4Ø only if the player wants to twist. The line sets the buy flag to 1—the buy flag is set to 1 if the player has twisted and therefore cannot buy any more cards, but remains at zero if the option to buy cards is still open. Line 1Ø5Ø calls PROCTWIST to turn the card over, add to the totals and perform the checks—see earlier. The program returns to Line 94Ø. Notice that PROCRESPONSE only ends when the player chooses to stick, or busts.

Having completed PROCRESPONSE, the program returns to Line 31Ø. The line is a check for pontoon. If the total is 21 and only two cards have been dealt, the player must be holding pontoon. Line 32Ø checks if E is 1, and tells the player FIVE-CARD TRICK if it is. The next check—in Line 48Ø—checks if the player has any chips left by checking if M is less than one. The program ends here.

The program also ends if the player has managed to accumulate more than 1,000 chips. Line 49Ø checks for this and tells the player YOU HAVE BUST THE BANK AND THIS TABLE IS CLOSING if the player has managed to come to this happy conclusion.

Lines 5ØØ to 55Ø are simply a variation on a 'Do you want another go?' routine. Line 5ØØ asks if the player wants another hand. If the game ends, Line 51Ø tells the player how many chips he has remaining. Line 52Ø makes sure that the player has replied with a Y or an N, and Line 55Ø starts a new deal if the player wants another hand.

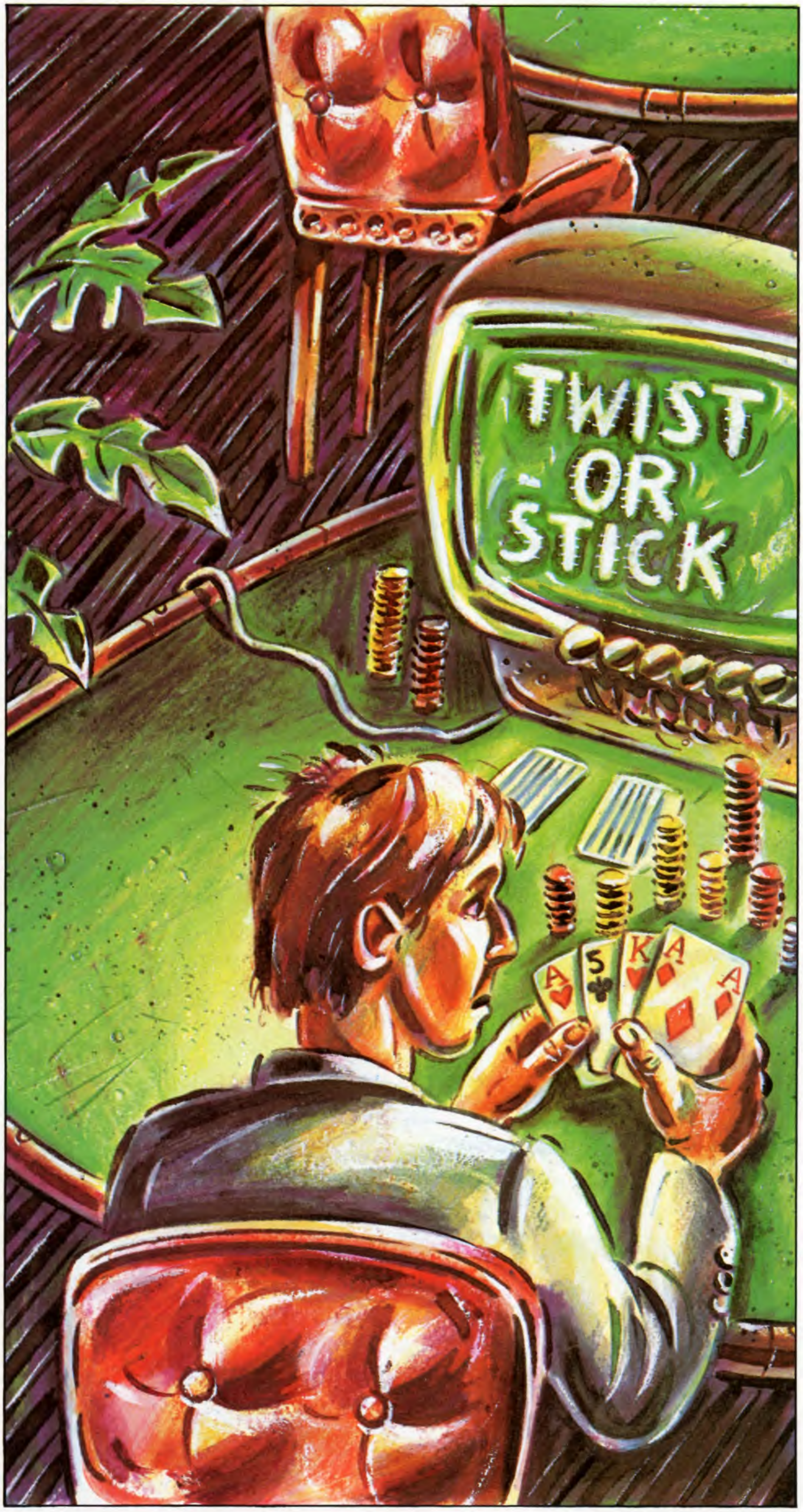
Now type in these lines which are in addition to those you have already entered:

```
170 MN=1ØØ
19Ø PCLS6
2ØØ CX=6:CY=11:PL=Ø:DL=Ø:
   NC=2:DA=Ø:PA=Ø:P5=Ø:
   TW=Ø:PF=Ø:NA=N
21Ø GOSUB3ØØØ
22Ø POKE178,156:D1=N:GOSUB
   1ØØØ:LINE(6,1Ø8)-(5Ø,18Ø),
   PSET,BF
23Ø FORK=1TO2ØØØ:NEXT
24Ø CLS:PRINT"□YOU HAVE";PL;:
   IF PA=1 THEN PRINT"OR 11"
   ELSE PRINT
25Ø PRINT:PRINT"□YOU HAVE";
   MN;"CHIPS LEFT"
26Ø INPUT"□WHAT IS YOUR BET□";BT
27Ø BT=INT(BT):IF BT<1 OR BT>MN
   THEN 24Ø
28Ø OB=BT:MN=MN-BT
29Ø CX=56:GOSUB3ØØØ:
   POKE178,156
3ØØ D2=N:GOSUB1ØØØ:LINE(56,1Ø8)
   -(1ØØ,18Ø),PSET,BF
31Ø FORK=1TO25ØØ:NEXT
32Ø IFPL=11 AND PA=1 ANDNC=2
   THEN65Ø
33Ø CX=CX+5Ø
34Ø PT=PL+1Ø*(PA AND(PL<12))
35Ø IFPL>21 THEN68Ø
36Ø CLS:IF NC=5 THEN PRINT"□YOU
   HAVE A 5-CARD TRICK":P5=1:
   FORK=1TO15ØØ:NEXT:GOTO 5ØØ
37Ø PRINT"□YOU HAVE";PL;:IF PA=1 AND
   PL<12THEN PRINT"OR";PT ELSE PRINT
38Ø PRINT:PRINT"□YOUR BET IS";BT
39Ø PRINT:PRINT"□YOU HAVE";
   MN;"CHIPS LEFT"
4ØØ PRINT:IF TW=Ø THENPRINT
   "□(B)UY OR";
41Ø PRINT"□(T)WIST";:IF PT>15 THEN
   PRINT"□OR (S)TICK";
42Ø PRINT"□?"
43Ø A$=INKEY$:IF A$<>"T" AND
   (A$<>"S"ORPT<16) AND
   (A$<>"B"ORTW=1) THEN 43Ø
44Ø IF A$="S" THEN 49Ø
45Ø IF A$="T" THEN TW=1:GOTO 48Ø
46Ø IF MN<OB THEN PRINT"□YOU
   HAVEN'T ENOUGH MONEY":GOTO 4ØØ
47Ø BT=BT+OB:MN=MN-OB
48Ø NC=NC+1:GOSUB3ØØØ:GOTO31Ø
49Ø IF PT<16 THEN PRINT"□YOU CAN'T
   STICK ON";PT:CX=CX-5Ø: GOTO 37Ø
5ØØ GOTO 75Ø
65Ø CLS:PRINT"□YOU HAVE A
   PONTOON":FORK=1TO1ØØØ:
   NEXT:PF=1:GOTO 19Ø
68Ø CLS:PRINT"□YOU'VE BUST AND LOSE
   YOUR BET"
75Ø PRINT:PRINT"□YOU HAVE";
   MN;"CHIPS LEFT":PRINT:IF
   MN>999 THENPRINT"□well done,
   you broke the bank":SCREENØ,1:
   PLAY"T6ADFBFEADC":GOTO79Ø
76Ø PRINT:PRINT"□PRESS (S) TO START
   AGAIN"
77Ø IFINKEY$<>"S" THEN77Ø
78Ø GOTO 19Ø
3ØØØ SCREEN1,1:GOSUB1ØØØ:
   GOSUB2ØØØ:IF NM>1Ø THEN
   PL=PL+1Ø ELSE PL=PL+NM
3Ø1Ø IF NM=1 THEN PA=1
3Ø2Ø RETURN
```

Line 17Ø sets the number of chips to 100. Line 19Ø clears and colours the screen.

Line 2ØØ initializes a number of variables: CX and CY are the coordinates of the top left-hand corner of the card, PL and DL are the player's score and dealer's score, NC is the number of cards being held by either the dealer or the player, DA and PA are flags which are set when the dealer or player is holding an ace, P5 is set when the player holds a five-card trick, TW is the twist flag indicating that the player has chosen to twist and can no longer buy cards, PF is the pontoon flag, and NA is the number of the first card dealt.

The subroutine between Lines 3ØØØ and 3Ø1Ø is called by Line 21Ø. A card is displayed on the screen, if it's greater than ten, the player's score is increased by ten, otherwise, the score is increased by the card's value. Line 3Ø1Ø checks if the card is an ace, and sets the player's ace flag if it is.



DEALING CARDS

RETURNing to the program, Line 22Ø uses a POKE to set up the stripey card backs. D1 is the dealer's first card; the subroutine works out its suit and number. LINE draws the card, and POKE makes the pattern.

There's a pause in Line 23Ø before Line 24Ø clears the text screen, and PRINTs the score. Using the PRINT command automatically switches from the high resolution screen to the text screen. The additional message OR 11 appears if the card is an ace. The number of chips in the player's possession is displayed by Line 25Ø. Line 26Ø looks after the player's bet, and Line 27Ø checks that the bet is larger than zero and within the player's resources—if it isn't, then the bet has to be made again. The stake is subtracted from the money held by the player in Line 28Ø.

The next stage is for Lines 29Ø and 3ØØ to call the dealing subroutines and display the back of the dealer's card.

Line 32Ø checks if the player has a pontoon—an ace and a ten or a picture—after Line 31Ø makes a short pause in the proceedings. If a pontoon is found, Line 65Ø PRINTs YOU HAVE A PONTOON, and starts again.

Line 33Ø looks after the position of the next card. PT is the higher score if the cards include an ace. PL is the lower score, or the only score, if there is no ace. If PL exceeds 21, then Line 68Ø tells the player YOU'VE BUST AND LOSE YOUR BET. The program continues to the messages in Line 75Ø onwards. Line 36Ø checks if the player has a five-card trick. The flag P5 is set, and there is a pause before continuing.

Line 37Ø displays the value of the cards the player is holding—two possibilities if the cards include an ace, and the lower value is less than 12. The player's bet is displayed by Line 38Ø and remaining chips by Line 39Ø.

RESPONSES

The BUY, TWIST or STICK prompts are displayed by Lines 4ØØ to 42Ø. Lines 43Ø to 45Ø look for the player's response to the prompts. If the player tries to buy cards when there isn't enough money, Line 46Ø responds. Line 47Ø adjusts the bet and the chip total, before another card is dealt by Line 48Ø.

Line 49Ø deals with the 'Stick' option. If PT is less than 16 the player is told YOU CAN'T STICK ON... The program continues to Line 75Ø for the moment, which tells the player the number of chips remaining, and checks if the bank has been broken. There's a sound after the colour set is changed. Lines 76Ø to 78Ø offer the player another go.

WIREFRAMES IN 3-D

So far, your wireframe drawing has been limited to simple lines and two-dimensional shapes. But put these together in the right way, and you can draw in three dimensions

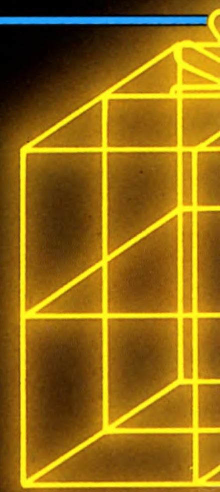
In theory, at least, it is nonsense to say that you can create a three-dimensional object (a box, say) on a two-dimensional medium—whether this is a piece of paper or a TV screen. But what you *can* do is to employ a visual convention so that our eyes perceive or 'read' a series of lines as a representation of a solid object.

VISUAL CONVENTIONS

A painting of a landscape, for example, may appear to have depth and distance because the artist has employed visual trickery—perspective. Perspective depends on the fact that we understand objects to get smaller as they move into the distance, and that parallel lines appear to converge as they get farther from the viewer.

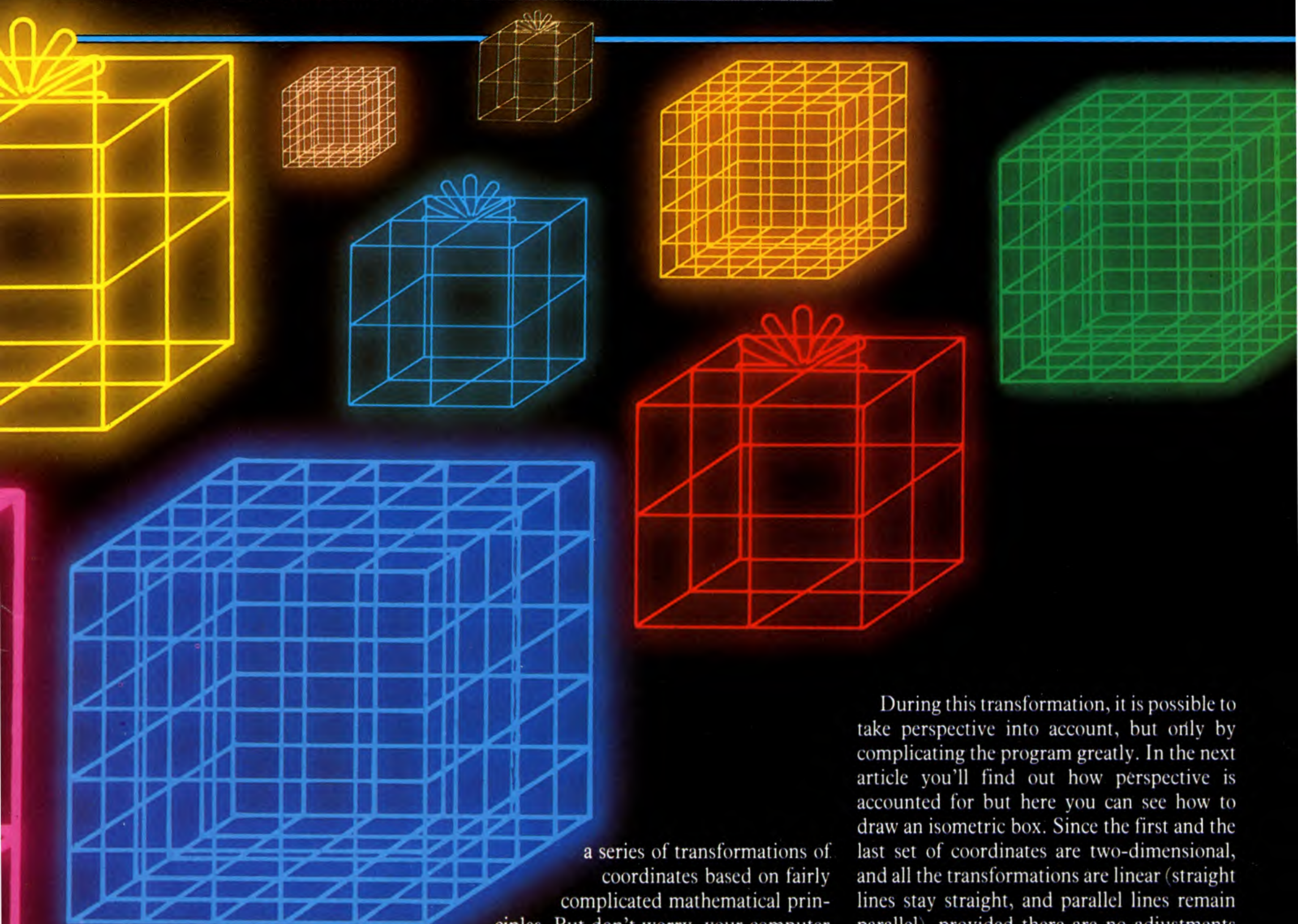
Perspective is not the only convention that is used for this type of representation. In technical drawing, it is common to use a convention called isometric projection. In this, lines which recede from the viewer are drawn at an angle (as in perspective drawing). Unlike perspective drawing, however, these lines do not converge, nor do objects get smaller as they get farther away. This has a number of advantages for technical applications—not least that you can take direct measurements of any line because its scale isn't affected by its direction.

Although you can represent perspective on a computer display (and a later article shows how to do this), this is harder to set up than an isometric projection. Isometric drawing relies simply on setting up a third axis on the screen. You already have an X axis (horizontal) and a Y axis (vertical). All you need now is a third





- DRAWING IN THREE DIMENSIONS
- ISOMETRIC PROJECTION
- TRANSFORMING THE COORDINATES
- JOINING GRIDS TO BUILD UP A CUBE
- USING THE PROGRAM TO DRAW OTHER SHAPES
- ADDING A BOW TO THE BOX



axis, the Z axis, which is at a set angle to the first two. Any line which is drawn at this angle is then understood to represent a line moving away from (or towards) the viewer. This is much simpler to understand if you look at the diagram of the axes (**fig. 1**).

TRANSFORMING THE PROGRAM

Using this convention, one way to produce three-dimensional wireframe images is to build them up from two-dimensional shapes, using routines like the one for the grid and circle given last time. Some of these two-dimensional shapes will need to be distorted to fit the angle of the Z axis. This requires a series of transformations of coordinates based on fairly complicated mathematical principles. But don't worry, your computer can handle them very easily.

The first thing you have to do is to transform the 2-D coordinates (X,Y) to 3-D coordinates (X′,Y′,Z′) in space in the desired plane. This means you have to specify how far along the Z axis the plane should be. Next you need to transform the 3-D coordinates (X′,Y′,Z′) to a new set of 3-D coordinates (Xe,Ye,Ze) based on the position and direction from which you are looking at the object. Finally, you have to transform these 3-D coordinates (Xe,Ye,Ze) back to the 2-D coordinates (Xp,Yp) so they can be displayed on the screen. Again, this is easier to follow if you look at a diagram (**fig. 2**), which shows a single shape being modified for six sides.

During this transformation, it is possible to take perspective into account, but only by complicating the program greatly. In the next article you'll find out how perspective is accounted for but here you can see how to draw an isometric box. Since the first and the last set of coordinates are two-dimensional, and all the transformations are linear (straight lines stay straight, and parallel lines remain parallel), provided there are no adjustments for perspective, you can achieve these transformations by changing the original Initialize and Draw routines.

If you saved a copy of the program from the first article, LOAD it, then key the lines below. It is a good idea to save the entire listing, because you will need to call some of the routines as the program develops in future articles. If you did not save the program, key Lines 5000 to 5130 (5000 to 5180 for Acorn micros) on page 512. These lines are the routine to draw a grid. Acorn users should also type in Lines 9500 to 9550 (the line-drawing routine) on page 511.

Now enter the following lines but do not run the program yet:

Delete Lines 15Ø and 155 then enter:

```
9ØØØ LET YX=Ø: LET XY=Ø: LET XX=1:
   LET YY=1: LET XO=Ø: LET YO=Ø
9Ø1Ø BORDER 4: PAPER 7: INK Ø: CLS
9Ø7Ø RETURN
91ØØ REM MOVE
911Ø PLOT XS*XX+XY*YS+XO+127,
   YX*XS+YY*YS+YO+76
912Ø RETURN
92ØØ REM DRAW
921Ø DRAW XE*XX+XY*YE+XO
   +127-PEEK 23677,YX*XE+YY*YE
   +YO+76-PEEK 23678
922Ø RETURN
95ØØ REM LINE
951Ø GOSUB 91ØØ
952Ø GOSUB 92ØØ
955Ø RETURN
```

```
9ØØØ PRINT "♥"
9Ø3Ø XX=1
9Ø4Ø YY=1
9Ø7Ø RETURN
95ØØ Q1=XS*XX+XY*YS+XO+16Ø:
   Q2=YX*XS+YY*YS+YO+1ØØ:
   Q3=XE*XX+XY*YE+XO+16Ø
95Ø1 Q4=YX*XE+YY*YE+YO+1ØØ:
   LINE Q1,Q2,Q3,Q4,1
955Ø RETURN
```

```
9ØØØ SCNCLR
9Ø3Ø XX=1
9Ø4Ø YY=1
9Ø7Ø RETURN
95ØØ Q1=XS*XX+XY*YS+XO+512:
   Q2=YX*XS+YY*YS+YO+512:
   Q3=XE*XX+XY*YE+XO+512
95Ø1 Q4=YX*XE+YY*YE+YO+512:
   DRAW 1,Q1,Q2 TO Q3,Q4
955Ø RETURN
```

```
9ØØØ DEF PROCINIT
9Ø1Ø CLS:CLG
9Ø2Ø XMAX=128Ø:YMAX=1Ø24
9Ø3Ø XMID=XMAX/2:YMID=YMAX/2
9Ø4Ø XX=1:XY=Ø:XO=Ø
9Ø5Ø YX=Ø:YY=1:YO=Ø
9Ø6Ø VDU29,XMID;YMID;
9Ø7Ø ENDPROC
911Ø MOVE FNSCREENX,FNSCREENY
921Ø DRAW FNSCREENX,FNSCREENY
93ØØ DEF FNSCREENX
931Ø =XX*X+XY*Y+XO
94ØØ DEF FNSCREENY
941Ø =YX*X+YY*Y+YO
```

```
9ØØØ PCLS
9Ø3Ø XX=1
9Ø4Ø YY=1
9Ø7Ø RETURN
95ØØ LINE(XS*XX+XY*YS+XO+127,
   YX*XS+YY*YS+YO+95)-(XE*XX
   +XY*YE+XO+127,YX*XE+YY*YE
   +YO+95),PSET
955Ø RETURN
```

The Initialize routine from Line 9ØØØ to Line 9Ø7Ø initializes the transformation variables (XX,XY,YX,YY,XO,YO) to values that make the drawing routine work as before. On the Dragon and Tandy micros, it is not essential to initialize variables before you use them, but it is, nonetheless, good practice, and it makes the program easier to understand.

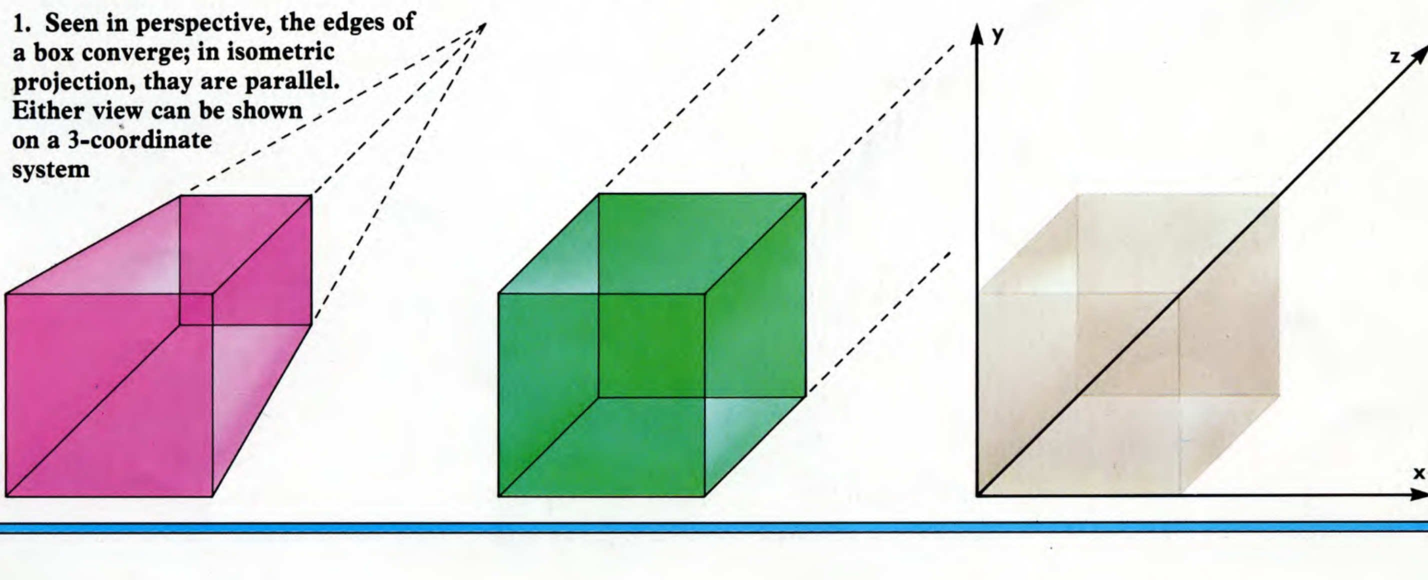
The routine moves the axes so that the origin is at the centre of the screen. On Acorn micros, this is achieved by a single command (Line 9Ø6Ø), whereas on the others, it is achived by adding an offset to the coordinates. The origin is moved because it is convenient to think of the viewer's eye as being directly above the origin at the centre of the screen—this point will become clearer in the next article when you see how to change your viewpoint and to add perspective.

To see the effect of each of the transformation variables, and to get a better feel for what is happening, change Lines 1ØØ to 18Ø as below to call the Grid routine:

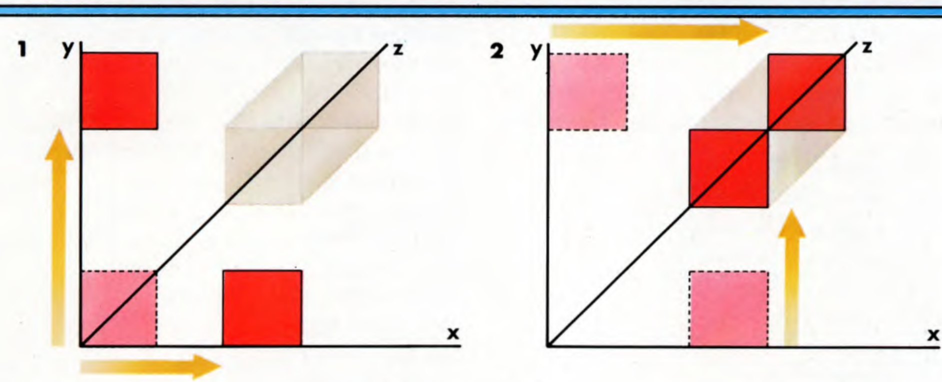
```
1ØØ GOSUB 9ØØØ
12Ø CLS
13Ø LET XA=-4Ø: LET YA=-2Ø: LET
   LW=4Ø: LET LH=4Ø: LET NX=5: LET
   NY=5
135 GOSUB 5ØØØ
14Ø LET XA=Ø: LET YA=-2Ø: LET
   LW=4Ø: LET LH=4Ø: LET NX=1Ø: LET
   NY=1Ø
145 GOSUB 5ØØØ
16Ø INPUT "ENTER XX,XY,YX,YY,
   XO,YO",XX,XY,YX,YY,XO,YO
17Ø GOTO 12Ø
18Ø STOP
```

Delete Line 155 then enter:

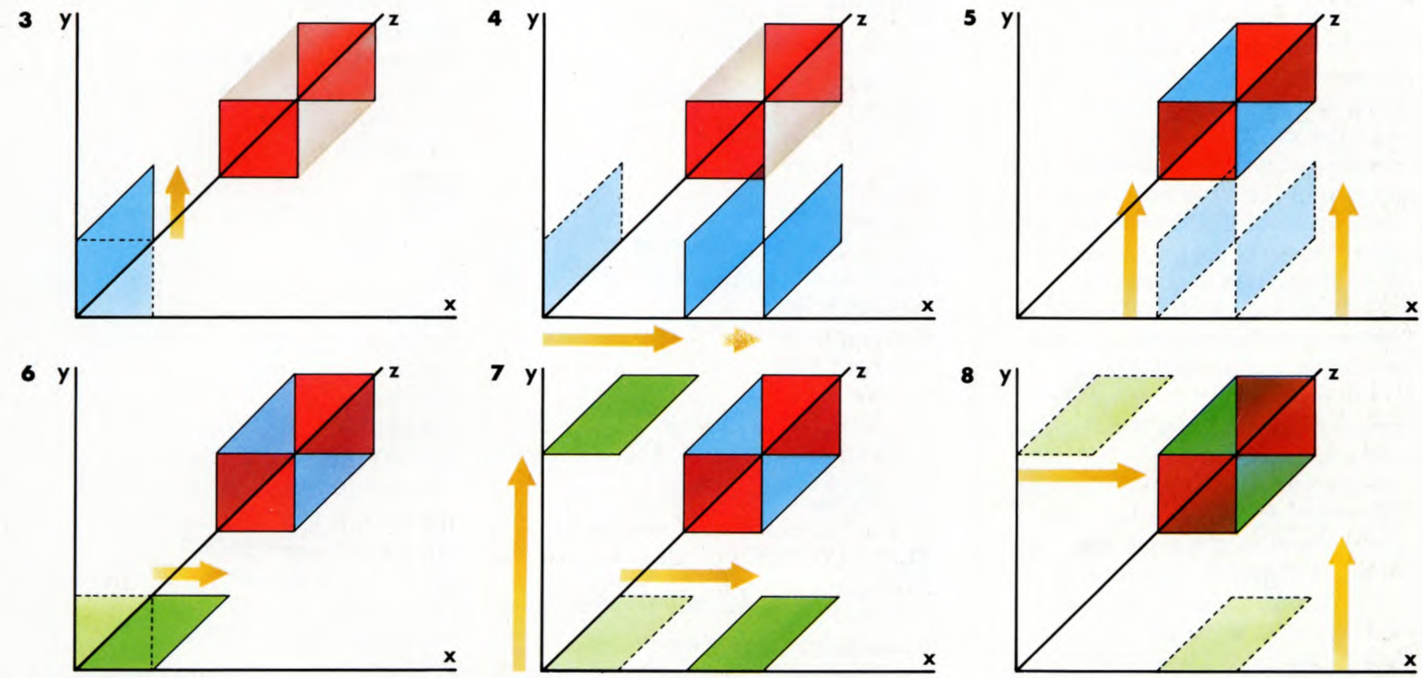
```
1ØØ COLOUR 5,1
11Ø GOSUB 9ØØØ
13Ø XA=-4Ø:YA=-2Ø:LW=4Ø:
   LH=4Ø:NX=5:NY=5
135 GOSUB 5ØØØ
14Ø XA=Ø:YA=-2Ø:LW=4Ø:
   LH=4Ø:NX=1Ø:NY=1Ø
145 GOSUB 5ØØØ
15Ø GET A$:IF A$="" THEN 15Ø
16Ø NRM:INPUT"ENTER XX,XY,YX,
   YY,XO,YO";XX,XY,YX,
   YY,XO,YO
17Ø HIRES Ø,1: GOTO 13Ø
```



1. Seen in perspective, the edges of a box converge; in isometric projection, thay are parallel. Either view can be shown on a 3-coordinate system



2. To depict a 3-D image on a 2-D plane, a square at the origin (1) can be shifted once along the X and Y axes then a second time (2) to form the front and back of the box. To form the sides, the square is sheared (3) in the Y direction then shifted (4) and (5) along the axes. Similarly, the top and bottom are formed by shearing in the X direction (6) then shifting it (7) and (8)



[Commodore logo]

Delete Line 155 then enter:

```
100 COLOR 1,5,Ø,Ø
11Ø GOSUB 9ØØØ
13Ø XA=-2ØØ:YA=-1ØØ:LW=2ØØ:
   LH=2ØØ:NX=5:NY=5
135 GOSUB 5ØØØ
14Ø XA=Ø:YA=-1ØØ:LW=2ØØ:
   LH=2ØØ:NX=1Ø:NY=1Ø
145 GOSUB5ØØØ
15Ø GET A$:IF A$="" THEN 15Ø
16Ø GRAPHIC Ø:PRINT "ENTER XX,XY,YX,
   YY,XO,YO":INPUT XX,XY,YX,
   YY,XO,YO
17Ø GRAPHIC 2: GOTO 13Ø
```

Also change Line 511Ø to:

```
511Ø YS=YS-JA
```

[BBC logo]

```
1ØØ MODE Ø
11Ø PROCINIT
12Ø REPEAT
13Ø CLS
14Ø PROCGRID(-2ØØ,-1ØØ,2ØØ,
   2ØØ,5,5)
15Ø PROCGRID(Ø,-1ØØ,2ØØ,2ØØ,
   1Ø,1Ø)
16Ø INPUT"ENTER XX,XY,YX,YY,
   XO,YO□",XX,XY,YX,YY,XO,YO
17Ø UNTIL FALSE
```

[Dragon/Tandy logo]

```
1ØØ PMODE4:SCREEN1,1
1Ø5 PI=4*ATN(1)
11Ø GOSUB 9ØØØ
12Ø CLS:PCLS:SCREEN1,1
13Ø XA=-4Ø:YA=-2Ø:LW=4Ø:
   LH=4Ø:NX=5:NY=5
135 GOSUB 5ØØØ
14Ø XA=Ø:YA=-2Ø:LW=4Ø:LH=4Ø:
   NX=1Ø:NY=1Ø
145 GOSUB 5ØØØ
15Ø A$=INKEY$:IF A$=""
   THEN 15Ø
16Ø INPUT"ENTER XX,XY,YX,YY,
   XO,YO□";XX,XY,YX,YY,XO,YO
17Ø GOTO 12Ø
```

RUN the program to see two small square grids at the centre of the screen, with a line asking you to enter XX,XY,YX and so on (Commodore users need to press a key). These are the transformation variables.

XX and YY set scale factors in the horizontal and vertical directions. If they are given negative values, then a reflection occurs about the axis. XO and YO set offsets for positioning the grids at different positions on the screen. XY and YX allow rotations and distortions (shearing) of the image. The transform variables have initial values XX=1, XY=Ø, YX=Ø, YY=1, XO=Ø and YO=Ø. To transform the image, you need to give these variables different values—this is the reason for the INPUT statement at Line 16Ø. Here is a set of values for you to try; enter them separated by a comma as they are printed, then press ENTER or RETURN; (or press ENTER after each one on the Spectrum):

-1,Ø,Ø,1,Ø,Ø causes reflection about the centre vertical axis of the screen.

Ø.5,Ø,Ø,2,Ø,Ø scales the image by two vertically and by a half horizontally.

Q+A

Can I change this program to make it draw different images?

The basic routine in this program is the one that draws the grid. It uses the section that draws lines, which you could use in a new program of your own to draw triangles, say. This would not differ greatly from the Grid routine, but you would need a little more care if you tried to call it several times to draw a shape, such as a tetrahedron—a four-sided figure with triangular sides. To draw the base, the values given to the transformation variables would be as for the box, but then you would have to abort the drawing of the side you do not need—the top of the box. A much more demanding task is to change the transformation variables (Lines 27Ø to 3ØØ) to angle the triangular grid and position it to draw the three sides.

It is much simpler to change the program to make it draw different sized boxes, and unclosed shapes. For example, change the value of L at Line 12Ø to vary the size of the box, then change the value of N at the same line. At large values of N, the box is virtually solid. To abort the drawing of the front and back sides, delete Line 311 (which later draws a bow on the box) and insert:

155 GOTO 22Ø, then RUN.

1,Ø,Ø,1,1ØØ,−5Ø moves the image 1ØØ units to the right and 5Ø units downwards or upwards on the Commodores, Dragon and Tandy. For a more noticeable change, Acorn users should try 1,Ø,Ø,1,4ØØ,−2ØØ, and Spectrum users 1,Ø,Ø,1,5Ø,−4Ø.
Ø,−1,1,Ø,Ø,Ø rotates the image 90 degrees anticlockwise about the centre of the screen.
1,Ø.5,Ø,1,Ø,Ø shears the image horizontally.
1,Ø.5,Ø.5,1,Ø,Ø shears it both horizontally and vertically.
All except Acorn users should press SPACE after each image is drawn.

To be able to predict the results accurately, you really need to understand matrix arithmetic. So although you can vary the shape of this image, you might not know how to combine values and grids to make a complete 3-D image. This is just what the next program does, however, using the procedures above.

Delete Lines 135, 145 and 175 from the previous program before you key this section:

```
11Ø GOSUB 9ØØØ
12Ø LET L=1Ø8: LET N=4
13Ø LET DX=.45*L: LET DY=.3*L
14Ø LET XN=−(L+DX)/2
15Ø LET YN=−(L+DY)/2
16Ø LET XO=XN
17Ø LET YO=YN
18Ø GOSUB 5ØØ
19Ø LET XO=XN+DX
2ØØ LET YO=YN+DY
21Ø GOSUB 5ØØ
22Ø LET XY=DX/L: LET XO=XN
23Ø LET YY=DY/L:LET YO=YN
24Ø GOSUB 5ØØ
25Ø LET YO=YN+L
26Ø GOSUB 5ØØ
27Ø LET XX=DX/L: LET XY=Ø
28Ø LET YX=DY/L: LET YY=1: LET YO=YN
29Ø GOSUB 5ØØ
3ØØ LET XO=XN+L
31Ø GOSUB 5ØØ
32Ø STOP
5ØØ LET XA=Ø: LET YA=Ø: LET LW=L:
    LET LH=L: LET NX=N: LET NY=N
51Ø GOSUB 5ØØØ
52Ø RETURN
```

Delete Lines 135, 145, and 175, then enter:

```
1ØØ HIRES Ø,1:COLOUR 5,1
```

Followed by Lines 11Ø to 52Ø, below.

Delete Lines 135,145, and 175, then enter:

```
1ØØ GRAPHIC 2:COLOR 1,5,Ø,Ø
```

Followed by Lines 11Ø to 52Ø, below.

Enter:

```
1ØØ PMODE4:SCREEN1,1
```

Followed by Lines 11Ø to 52Ø, below.

Delete Lines 135 and 145 first then enter:

```
11Ø GOSUB 9ØØØ
12Ø L=12Ø:N=5
13Ø DX=.45*L:DY=.3*L
14Ø XN=−(L+DX)/2
15Ø YN=−(L+DY)/2
16Ø XO=XN
17Ø YO=YN
18Ø GOSUB 5ØØ
19Ø XO=XN+DX
2ØØ YO=YN+DY
21Ø GOSUB 5ØØ
22Ø XY=DX/L:XO=XN
23Ø YY=DY/L:YO=YN
24Ø GOSUB 5ØØ
25Ø YO=YN+L
26Ø GOSUB 5ØØ
27Ø XX=DX/L:XY=Ø
28Ø YX=DY/L:YY=1:YO=YN
29Ø GOSUB 5ØØ
3ØØ XO=XN+L
31Ø GOSUB 5ØØ
32Ø GOTO32Ø
5ØØ XA=Ø:YA=Ø:LW=L:LH=L:
    NX=N:NY=N
51Ø GOSUB 5ØØØ
52Ø RETURN
```

```
1ØØ MODEØ
11Ø PROCINIT
12Ø L=6ØØ:N=5
13Ø DX=Ø.45*L:DY=Ø.3*L
14Ø XN=−(L+DX)/2
15Ø YN=−(L+DY)/2
16Ø XO=XN
17Ø YO=YN
18Ø PROCSIDE:REM FRONT
19Ø XO=XN+DX
2ØØ YO=YN+DY
21Ø PROCSIDE:REM BACK
22Ø XY=DX/L:XO=XN
23Ø YY=DY/L:YO=YN
24Ø PROCSIDE:REM BOTTOM
25Ø YO=YN+L
26Ø PROCSIDE:REM TOP
27Ø XX=DX/L:XY=Ø
28Ø YX=DY/L: YY=1: YO=YN
29Ø PROCSIDE:REM LEFT
3ØØ XO=XN+L
31Ø PROCSIDE:REM RIGHT
32Ø END
5ØØ DEF PROCSIDE
51Ø PROCGRID(Ø,Ø,L,L,N,N)
52Ø ENDPROC
```

When you RUN the program above, you should see a cube at the centre of the screen. The sides are all drawn by the same routine, which is simply the one to draw a grid with suitable variables to transform it into the required shape. Two other variables have to be set up before the cube can be drawn. These are the variable L which sets the size of the cube, and N which sets the number of grid squares. The variable DX specifies how far to the left or right the back of the cube is displaced from the front, and DY specifies how far to the top or bottom the same two sides are displaced. DX and DY also ensure the cube appears as wide as it is thick.

Another pair of variables—XN at Line 140 and YN at Line 150—set offsets for the starting position of the front of the cube. These offsets are passed to XO at Line 160 and YO at Line 170, then the back of the cube is drawn (at Line 180). (On the Acorns the order of drawing is reversed.)

The front and back of the cube are identical, so to draw the front of the cube the only requirement is to increase the offset in the X-direction (Line 190) and the Y-direction (Line 200). These offsets align the position of the back with the sides you have drawn.

Line 240 draws the top, but first the grid must be transformed—XY and YY—and the offsets redefined, all at Lines 220 and 230. The same view of the grid is used for the bottom, which needs only be scaled and positioned (Line 250) before being drawn. Similarly, Lines 270 to 310 transform the grid once to draw the right side, then reposition it to draw the left side.

All the grid squares on each side are the same shape and size, and all the parallel lines in each grid remain parallel. The reason is that no adjustment has yet been made for perspective. Moreover, it is difficult to adapt the program to show other views of the cube.

The next article shows how to make perspective wireframe drawings which can be viewed from any position. In the meantime, change Lines 120 and 130 as below, and add these few lines to wrap up the cube as a present. Don't forget to SAVE a copy:

```
120 LET L=60: LET N=2
130 LET DX=.3*L: LET DY=.4*L
311 GOSUB 9920
9920 FOR M=0 TO PI STEP .01
9930 LET D=COS (6*M)
9940 LET S=D*COS M: LET T=D*SIN M
9950 LET S=S*20: LET T=ABS (T*20)
9960 PLOT S+127,T+106
9970 NEXT M
9980 RETURN
```

```
120 L=60:N=2
130 DX=.6*L:DY=.5*L
311 GOSUB 10000
10000 SS=160:TT=70
10010 FORM=0 TO π STEP.02
10020 D=COS(6*M)
10030 S=D*COS(M):T=D*SIN(M)
10040 S=S*24:T=ABS(T*20)
10050 LINE SS,TT,160+S,70-T,1
10055 SS=160+S:TT=70-T
10060 NEXT M
10130 RETURN
```

```
120 L=300:N=2
130 DX=.6*L:DY=.5*L
311 GOSUB 10000
10000 SS=512:TT=350
10010 FORM=0 TO π STEP.02
10020 D=COS(6*M)
10030 S=D*COS(M):T=D*SIN(M)
10040 S=S*150:T=ABS(T*250)
10050 DRAW 1,SS,TT TO 512+S,
  350-T
10055 SS=512+S:TT=350-T
10060 NEXT M
10070 RETURN
```

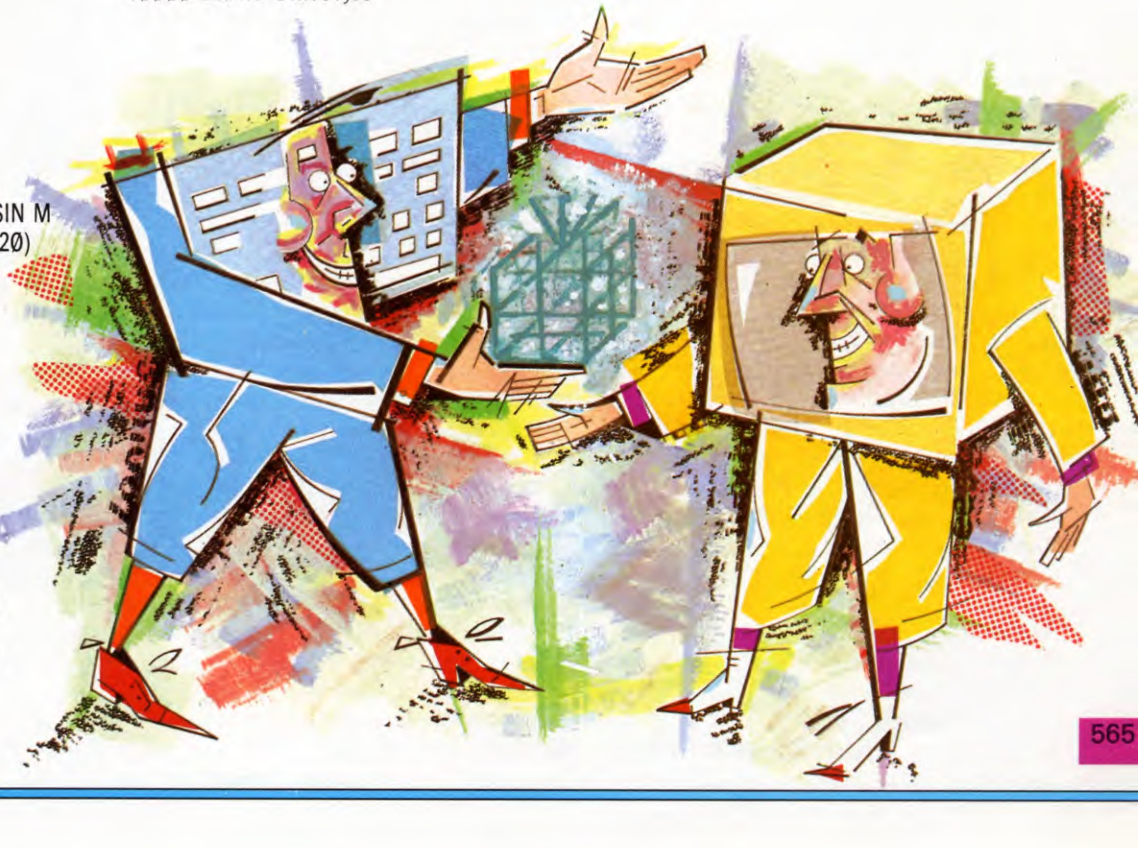
```
120 L=300:N=2
130 DX=0.6*L:DY=0.5*L
315 PROCPOLAR
10000 DEF PROCPOLAR
10010 MOVE120,148
10070 FOR M=0 TO PI STEP.01
10080 D=COS(6*M)
10090 S=D*COS(M):T=D*SIN(M)
10100 S=S*120:T=ABS(T*100)
10110 DRAW S,T+148
10120 NEXT
10130 ENDPROC
```

```
120 L=60:N=2
130 DX=.6*L:DY=.5*L
311 GOSUB 10000
10000 DRAW"BM151,65"
10010 FORM=0 TO 4*ATN(1) STEP .02
10020 D=COS(6*M)
10030 S=D*COS(M):T=D*SIN(M)
10040 S=S*24:T=ABS(T*20)
10050 LINE-(S+127,65-T),PSET
10060 NEXT
10130 RETURN
```

Microtip

Off-screen Drawing

If you change Line 120 so that L has large values, you might find that the program crashes, giving an error message. This is because some computers cannot draw a line if one of the points specified is off the screen. It does not happen on Acorn micros, which can draw any section of line and ignore the off-screen section. Provided the image is entirely on the screen, other users should have no problem. The program is not limited by this problem, because in the next article, a check is made to ensure that the machine tries to only draw lines that are valid. Because of this, the program is not interrupted with error messages.



COMPUTER AIDED DESIGN

Accurate, high-resolution drawing with fingertip control—that's the benefit of computer aided design. Enter this listing for easy graphics without writing extra programs

Given the right kind of software, your computer's graphics capabilities can be used to give you a powerful new drawing aid. Whether you are a skilled draughtsperson, or just like to doodle, you will be able to use the listing given here with the same—or even greater—freedom that pencil and paper can afford.

In industry, Computer Aided Design (CAD) software and mainframe computers let their users draw an object in detail and simulate how a particular shape responds to stresses due, for example, to structural loading, wind, vibration or changes in temperature. Such an exercise is well beyond the capacity of the home micro, for it requires a vast amount of data processing, as well as the best graphics capabilities, to produce different views of the subject and perform the calculations.

Often, however, a designer needs only to study either a detail or the overall view for its visual appearance. In this application, CAD becomes a tool to replace pencil and paper for technical drawing—allowing the accurate generation of curves, straight lines and geometric shapes, with the facility for instant correction or erasure.

It is this aspect of CAD which can be applied to home computers—whether you are interested in the technical aspects of design, or just want it for casual sketching. Although all the home computers (with the exception of the Commodores) already have drawing commands as part of their standard BASIC, it requires lengthy programming to construct a detailed picture. You have already seen examples of how this is done, in the articles on pages 84, 184 and 366.

The beauty of this design program is that once you have entered it, you can create any picture which is within your computer's high resolution graphics capability—without further programming. It puts all the drawing commands under simple, direct keyboard control—so that all you need do is to select an option or move a cursor using the cursor keys. And it also in some cases programs in extra commands which are not available in the machines' standard BASIC.

As the graphics capabilities of different computers vary widely, there are differences between the individual programs, each of which is designed to exploit the strengths and minimize the weaknesses of a particular machine. For example, some micros, such as the Spectrum and Acorns, can display text and graphics easily on the same screen whereas others, like the Dragon and Tandy, cannot. On the other hand, the Spectrum, Dragon and Tandy can store information other than that currently on view, in several 'screens', but the Acorns can store only a few.

Because of its limited graphics ability, there is no listing for the ZX81. And to access the high-res graphics from BASIC, Commodore 64 users need to plug in a Simons' BASIC cartridge, while Vic 20 users need a Super Expander cartridge. The Spectrum program is for a 48K machine.

DRAWING AIDS

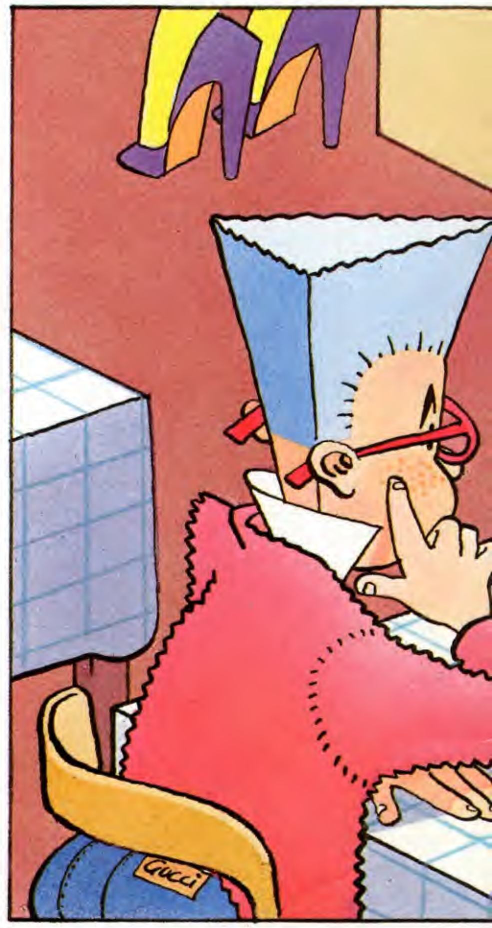
Despite individual differences, each program works in much the same way. The user is given a menu, or list, of drawing options for things like lines, ellipses, circles or rectangles. By selecting one of these and then using the cursor keys to position it on screen, you can build up some impressive drawings—with the bonus that all your lines will be straight and all your curves will be regular. And if you should make a mistake, some computers also give you the option to cancel what you've drawn and have another go.

There are also options to allow you to alter the drawing colour, or, on some to fill in areas from the palette. And when you have finished the picture, you can decide whether to clear it and start again, or to SAVE the image so it can be redrawn later. This facility also allows you to LOAD in an image that was not drawn with this program. You could for example take a favourite title page and modify it as you wish by redrawing certain areas.

USING THE PROGRAM

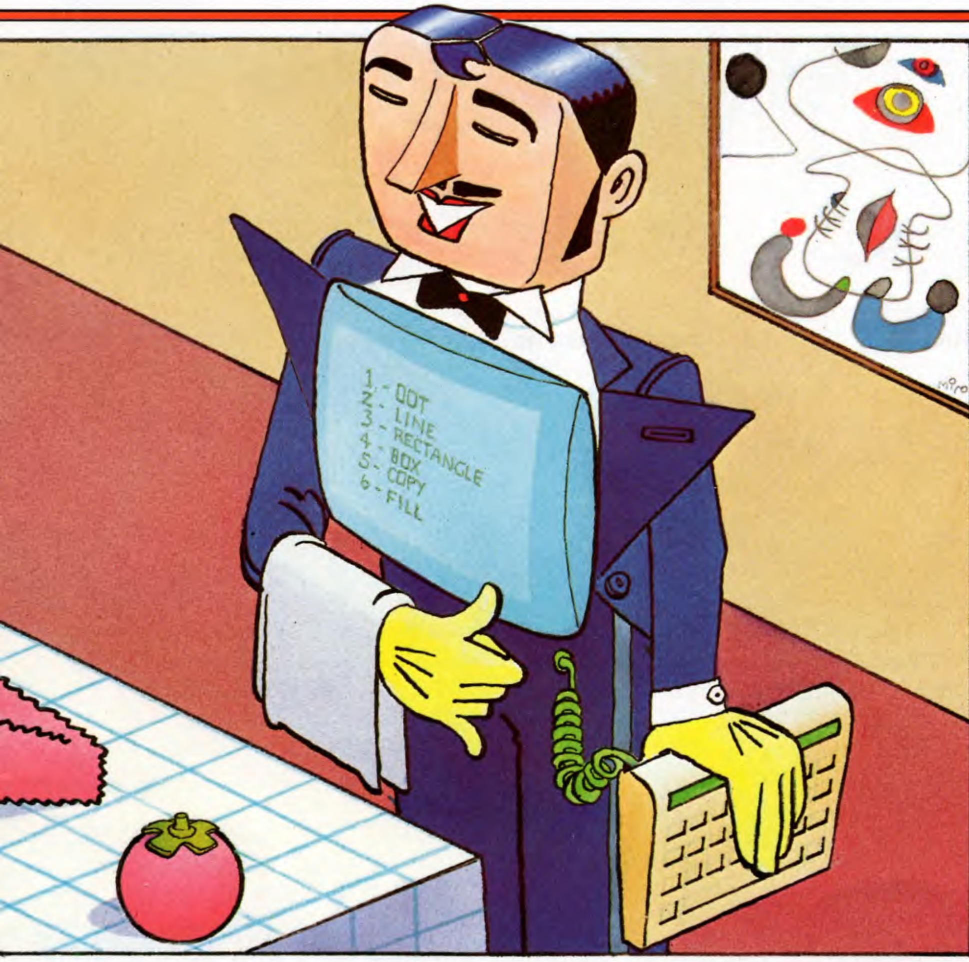
There are individual differences between each of the machines, which are explained in the detailed notes on each computer.

In each case (except for the Vic 20), the listings are in two parts, one given here, and one in the following article. The first part is designed so that it will run on its own and allow you to create line drawings like those shown here. But some of the more sophisticated options on the menu will not be added until later. The use of the Super Expander on the Vic 20 allows its listing to be much shorter than that for the other machines, so it is given here in complete form.



When you RUN the program, a menu and a cursor are displayed on the screen. To select an item from the menu, position the cursor to the left of the item, using the control keys Q (up), A (down), O (left) and P (right). Then press ENTER to make the selection, when the menu will clear, leaving the cursor on a blank screen. At any stage, you can return to, or leave, the menu by pressing ENTER.

- USING THE COMPUTER AS A DRAWING AID
- EXTENDING THE BUILT IN GRAPHICS COMMANDS
- USING THE PROGRAM
- SKETCHING AND FREEHAND DRAWING
- USING THE LINE OPTION FOR GREATER ACCURACY
- INTRODUCING COLOUR



The first item on the menu is Draw, which draws dots on the screen. To use this option, move the cursor to the point where you wish to start drawing, then press SPACE. When you next move the cursor, it will leave a trail of dots in whatever direction you move. You can speed up the movement by holding down SHIFT while pressing the control key to move. To exit the Draw option—when you have come to the end of the dotted line—press SPACE again, when you will be able to move the cursor without marking the screen, or if you like, return to the menu to select another option.

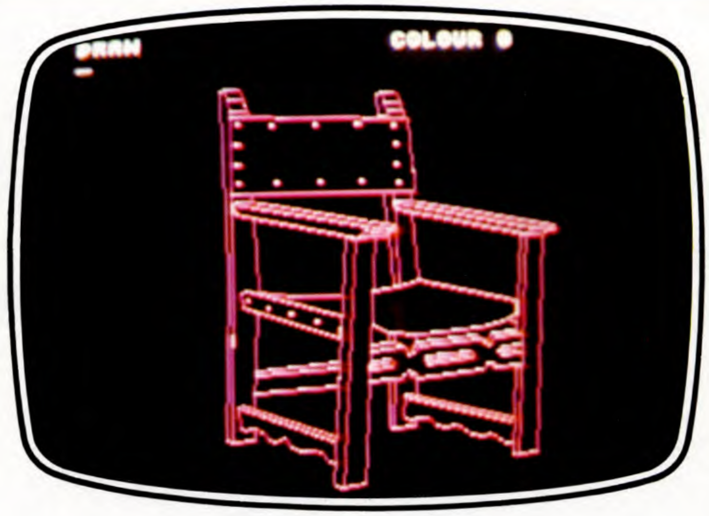
The Line option works just as Draw, except that it leaves a solid or continuous line on the screen; press ENTER to select it, move to the start and press SPACE, then move to the end and press SPACE again. The option you have selected will remain available, until you cancel it by selecting another. So you could move to another point and draw another line, and so on.

There are two options for colouring the screen—Paper lets you colour the border and background, and Ink lets you colour as you draw. When you select either of these options, prompts appear on the screen to help you select colours. The cursor shows the current ink colour, so you can see the effect of your selection instantly. When you have responded to all the prompts, the display returns to the menu, but the drawing cursor remains where you left it, so it is a simple matter to continue drawing. You can select Ink colours either before or while you draw. Only when you have exited the drawing option (by pressing SPACE a second time) will the line be fixed.

Warning:
This program contains machine code, so be sure to SAVE it before RUNning it:

```
10 BORDER 4: PAPER 7: INK Ø: OVER Ø: CLS
20 POKE 23658,Ø: LET OP=1
40 DIM O(12): FOR N=1 TO 12: READ O(N):
   NEXT N: LET x=127: LET y=87
60 LET x1=x: LET y1=y
65 LET xx=Ø: LET yy=Ø
70 FOR n=65368 TO 65368+73: READ a:
   POKE n,a: NEXT n
100 RANDOMIZE USR 65380
1020 PRINT INK 9;AT 2,9;
   "□□□□□□□□□□□□□□"
1030 PRINT INK 9;AT 3,9;
   "□□□DRAW□□□□□□□"
1040 PRINT INK 9;AT 4,9;
   "□□□LINE□□□□□□□"
1050 PRINT INK 9;AT 5,9;
   "□□□PAPER□□□□□□"
1060 PRINT INK 9;AT 6,9;
   "□□□INK□□□□□□□□"
1070 PRINT INK 9;AT 7,9;
   "□□□RECTANGLE□□"
1080 PRINT INK 9;AT 8,9;
   "□□□BOX□□□□□□□□"
1090 PRINT INK 9;AT 9,9;
   "□□□CIRCLE□□□□□"
1100 PRINT INK 9;AT 1Ø,9;
   "□□□ERASE□□□□□□"
1110 PRINT INK 9;AT 11,9;
   "□□□OOPS□□□□□□□"
1115 PRINT INK 9;AT 12,9;
   "□□□COPY□□□□□□□"
1120 PRINT INK 9;AT 13,9;
   "□□□LOAD□□□□□□□"
1130 PRINT INK 9;AT 14,9;
   "□□□SAVE□□□□□□□"
1140 PRINT INK 9;AT 15,9;
   "□□□□□□□□□□□□□□"
1150 PLOT 72,48: DRAW INK 9;111,Ø: DRAW
   INK 9;Ø,111: DRAW INK 9;-111,Ø: DRAW
   INK 9;Ø,-111
1155 FOR n=1 TO 1ØØ: NEXT n
1160 PRINT INVERSE 1; PAPER 9;AT
   OP+2,1Ø;">"
1170 PAUSE Ø: LET A$=INKEY$: IF A$=""
   THEN GOTO 117Ø
1175 IF A$=CHR$ 13 AND OP=9 THEN
   RANDOMIZE USR 654Ø4: RANDOMIZE
```





Outline drawing, as on the Spectrum and BBC B.... ....can be done with the listings given here

```
    USR 65368: GOTO 1ØØØ
118Ø IF A$=CHR$ 13 THEN RANDOMIZE
    USR 65392: RANDOMIZE USR 65368: FOR
    n=1 TO 1ØØ: NEXT n: GOTO O(OP)
119Ø PRINT AT OP+2,1Ø;"□"
12ØØ IF A$="a" THEN LET OP=OP+1: IF
    OP=13 THEN LET OP=12
121Ø IF A$="q" THEN LET OP=OP-1: IF
    OP=Ø THEN LET OP=1
122Ø GOTO 116Ø
2ØØØ REM draw
2ØØ5 FOR n=1 TO 5Ø: NEXT n
2Ø1Ø GOSUB 8ØØØ
2Ø6Ø IF INKEY$=CHR$ 13 THEN
    RANDOMIZE USR 6538Ø: GOTO 1ØØØ
2Ø7Ø IF INKEY$<>CHR$ 32 THEN GOTO
    2Ø1Ø
2Ø8Ø FOR n=1 TO 5Ø: NEXT n
2Ø9Ø GOSUB 8ØØØ: PLOT x,y
2Ø95 IF INKEY$=CHR$ 13 THEN
    RANDOMIZE USR 6538Ø: GOTO 1ØØØ
21ØØ IF INKEY$=CHR$ 32 THEN GOTO 2ØØ5
211Ø GOTO 2Ø9Ø
25ØØ REM line
25Ø5 FOR n=1 TO 5Ø: NEXT n
251Ø GOSUB 8ØØØ
2515 IF INKEY$=CHR$ 13 THEN
    RANDOMIZE USR 6538Ø: GOTO 1ØØØ
252Ø IF INKEY$<>CHR$ 32 THEN GOTO
    251Ø
2525 FOR n=1 TO 5Ø: NEXT n
253Ø LET xx=Ø: LET yy=Ø: LET hx=x: LET
    hy=y
254Ø GOSUB 8ØØØ: PLOT hx,hy: DRAW OVER
    1;xx,yy: FOR n=1 TO 5: NEXT n: PLOT
    hx,hy: DRAW OVER 1;xx,yy
255Ø IF INKEY$<>CHR$ 32 THEN GOTO
    254Ø
256Ø PLOT hx,hy: DRAW xx,yy: GOTO 25ØØ
3ØØØ REM paper & border
3Ø1Ø PRINT #1;AT Ø,Ø;"Paper colour (Ø to
    7)?"
3Ø15 LET a$=INKEY$
3Ø2Ø IF a$<"Ø" OR a$>"7" THEN GOTO
    3Ø15
3Ø3Ø POKE 65535,VAL a$*8
3Ø35 RANDOMIZE USR 65416
3Ø4Ø FOR n=1 TO 5Ø: NEXT n
3Ø5Ø PRINT #1;AT Ø,Ø;"Border colour (Ø to
    7)?"
3Ø6Ø LET a$=INKEY$
3Ø7Ø IF a$<"Ø" OR a$>"7" THEN GOTO
    3Ø6Ø
3Ø8Ø BORDER VAL a$
3Ø9Ø PRINT #1;AT Ø,Ø;"□";TAB
    31;"□□";TAB 31;"□"
3Ø95 RANDOMIZE USR 65416: RANDOMIZE
    USR 6538Ø: GOTO 1ØØØ
35ØØ REM ink
351Ø PRINT #1;AT Ø,Ø;"Select ink (Ø to 7)"
352Ø LET a$=INKEY$: IF a$<"Ø" OR
    a$>"7" THEN GOTO 352Ø
353Ø INK VAL a$
354Ø PRINT #1;AT Ø,Ø;"□";TAB 31;"□":
    GOTO 1ØØØ
75ØØ GOTO 1ØØØ
8ØØØ REM INKEY$ ROUTINE
8ØØ5 PLOT OVER 1;x,y
8Ø1Ø LET A$=INKEY$
8Ø2Ø IF A$="q" AND y<175 THEN LET
    y1=y+1: LET yy=yy+1
8Ø3Ø IF A$="a" AND y>Ø THEN LET
    y1=y-1: LET yy=yy-1
8Ø4Ø IF A$="p" AND x<255 THEN LET
    x1=x+1: LET xx=xx+1
8Ø5Ø IF A$="o" AND x>Ø THEN LET
    x1=x-1: LET xx=xx-1
8Ø6Ø IF A$="Q" AND y<172 THEN LET
    y1=y+4: LET yy=yy+4
8Ø7Ø IF A$="A" AND y>3 THEN LET
    y1=y-4: LET yy=yy-4
8Ø8Ø IF A$="P" AND x<252 THEN LET
    x1=x+4: LET xx=xx+4
8Ø9Ø IF A$="O" AND x>3 THEN LET
    x1=x-4: LET xx=xx-4
8Ø95 PLOT OVER 1;x,y
81ØØ LET x=x1: LET y=y1: RETURN
9ØØØ DATA 2ØØØ,25ØØ,3ØØØ,35ØØ,4ØØØ,
    4Ø2Ø,5ØØØ,55ØØ,Ø,6ØØØ,7ØØØ,75ØØ
9Ø1Ø DATA 17,Ø,64,33,8Ø,195,1,Ø,27,237,
    176,2Ø1,17,8Ø,195,33,Ø,64,1,Ø,27,
    237,176,2Ø1
9Ø2Ø DATA 17,168,222,33,8Ø,195,1,Ø,27,
    237,176,2Ø1,17,8Ø,195,33,168,222,
    1,Ø,27,237,176,2Ø1
9Ø3Ø DATA 33,Ø,88,6,4,197,6,176,2Ø3,158,
    2Ø3,166,2Ø3,174,58,255,255,134,119,
    35,16,242,193,16,236,2Ø1
```

A flashing cursor appears at the centre of the screen when you RUN this program. To move it about the screen, use the cursor control keys. Press D to select the Draw mode, then select a colour in which to draw, by pressing a key between 1 (the background) and 4. Now you can draw in any direction, by pressing the cursor keys. To 'lift up' the cursor temporarily, press 5—the border turns grey—then the cursor keys can be used to move without drawing. To start drawing again, press the arrow key at the top, left-hand corner of the keyboard to fix the starting position. This arrow key has the same function in any drawing mode. Next select a colour (1 to 4), and continue to draw, using the cursor keys. To exit any of the drawing modes, press *, which has exactly the same effect of selecting Move.

You can speed up the movement of the





Next time, you'll see how to add colour.... ....as in these Dragon and Commodore examples

cursor by pressing a key between 6 (normal speed) and 9 (fastest speed). This facility can be used to speed up the drawing rate, but it is not possible to draw to the edge of the screen when you use the faster speeds.

To select Line, press * to exit the current drawing mode, then move to the start of the line to be drawn and press L for Line, then move towards the end of the line. As you move the cursor, the line will flash between the starting position and the cursor, but it disappears when the cursor stops moving. If you want to see where the line is you can get it to flash again by pressing the space bar. To fix the line select a colour in which to draw, then press the space bar. Select a colour only when you are completely satisfied with the line or shape. Again, you can press 5 to lift the cursor and move to a new starting position, then fix the start by pressing the arrow key. If now you select a faster speed (key 8, say), you can move the cursor more quickly to any end position, and the line can be drawn faster—in long steps.

This program contains a section of machine code so, to be safe, you must SAVE it before you RUN it. Any error will cause the program to crash, so having a copy on tape or disk will save you having to type it in again.

```
10 POKE 51,255:POKE 52,94:
   POKE 55,255:POKE 56,94:CLR
20 IF FG=1 THEN 130
30 FOR Z=24320 TO 24431:READ X:
   POKE Z,X:NEXT Z:FG=1
40 DATA 169,0,141,14,220,169,53,133,1
50 DATA 169,0,133,251,133,253,169,224,
   133,252,169,96,133,254,160,0
60 DATA 177,251,145,253,192,63,208,16,
   165,252,201,255,208,10
70 DATA 162,1,142,14,220,162,55,134,1,
   96,200
80 DATA 208,229,230,252,230,254,76,25,95
90 DATA 165,45,133,253,165,46,133,254
100 DATA 162,8,160,1,169,1,32,186,255,
   162,113,160,95,173,112,95
110 DATA 32,189,255,169,0,133,251,169,
   96,133,252,162,0,160,128,169,251
120 DATA 32,216,255,165,253,133,45,165,
   254,133,46,96
130 HIRES 0,1:C(0)=1:C(1)=2:
   C(2)=5:C(3)=6:C(4)=12:
   MULTI C(1),C(2),C(3)
140 X=80:Y=100:SP=1:CO=4:
   F=0:GOTO 450
150 GET A$:FOR Z=1 TO 2:TEXT
   X,Y,"▧",4,1,1:NEXT Z:IF
   A$="" THEN 150
160 IF A$="◙" THEN PRINT "◙":
   NRM:HIRES 0,1:MULTI C(1),C(2),
   C(3):POKE 198,0
200 IF F=0 THEN CO=4
210 IF A$="←" THEN XX=X:YY=Y
220 IF A$="▌▌"ANDX-SP>0
   THENX=X-SP
230 IF A$="▙"ANDX+SP<159
   THENX=X+SP
240 IF A$="◘"ANDY-SP>0
   THENY=Y-SP
250 IF A$="▟"ANDY+SP<198
   THEN Y=Y+SP
270 IF VAL(A$)>0 AND VAL(A$)<6 THEN
   CO=VAL(A$)-1
290 IF VAL(A$)>5ANDVAL(A$)<10
   THEN SP=(VAL(A$)-6)*6+1
310 IF (F>0 AND F<5) OR F=6 THEN FOR
   Z=1 TO 2:LINE XX,YY,X,Y,4: NEXT Z
350 IF A$="L" THEN XX=X:YY=Y:
   F=3:CO=4
360 IF (F=3 AND A$="□")ANDCO<4
   THEN LINE XX,YY,X,Y,CO:F=0
390 IF A$="D" THEN F=5:XX=X:YY=Y
400 IF F=5 AND CO<4 THEN LINE
   XX,YY,X,Y,CO: XX=X:YY=Y
440 IF A$="*" THEN F=0
450 COLOUR C(CO),1
460 IF A$=CHR$(133) THEN GOSUB 520
470 IF A$=CHR$(134) THEN GOSUB 530
480 GOTO 150
520 POKE 24346,251:POKE 24348,253:
   SYS 24320:RETURN
530 POKE 24346,253:POKE 24348,251:
   SYS 24320:RETURN
```

The program for the Vic 20, with a Super Expander cartridge, is remarkably short for what it achieves—so the entire listing is given in this part of the article. RUN the program, when you should see a flashing point—the drawing cursor—on a blank screen. To move this cursor, use the control keys: Z (left), X (right), ';' (up) and '/' (down). This option lets you draw in any direction, but notice that the cursor moves in distinct steps. You can vary the length of the step by pressing a key between 5 (normal speed) and 9 (the fastest).

To move the cursor without drawing, hold down SHIFT while you press any of the cursor control keys. And by selecting a large step (key 8, say), you can speed up the movement of the cursor to where you wish to start drawing.

The colour in which you draw can be changed at any stage, by pressing a key between 1 (the background colour) and 4. Next time you can find out how to draw different shapes and to erase, using these functions.

```
10 GRAPHIC 1:C(1)=6:C(2)=2:
   C(3)=5:CO=1:SP=1:COLOR 1,
   C(1),C(2),C(3)
15 CC(6)=1:CC(2)=2:CC(5)=3:
   CC(1)=Ø:POKE 65Ø,128:
   LX=512:LY=512:PRINT CHR$(8)
2Ø X=512:Y=X:POINT Ø,X,Y:
   BC=Ø:GOTO 15Ø
1ØØ GET A$
1Ø5 IF VAL(A$)>Ø AND VAL(A$)<5
   THEN CO=VAL(A$)-1
1Ø6 IF A$>"4" AND A$<":" THEN
   SP=(VAL(A$)-5)*3Ø+1
11Ø IF PEEK(197)=33 AND X-SP>8 THEN
   X=X-SP
12Ø IF PEEK(197)=26 AND X+SP<1Ø23
   THEN X=X+SP
13Ø IF PEEK(197)=22 AND Y-SP>8 THEN
   Y=Y-SP
14Ø IF PEEK(197)=3Ø AND Y+SP<1Ø23
   THEN Y=Y+SP
15Ø IF A$="▢" THEN:SCNCLR
16Ø IF A$="F" THEN:PAINT CO,X-8,Y-8
17Ø IF PEEK(197)=34 THEN:
   CIRCLE CO,LX,LY,ABS(LX-X),
   ABS(LY-Y):POINT Ø,X,Y
18Ø IF PEEK(197)=8 THEN LX=X:LY=Y
185 IF PEEK(197)=15 THEN:
   POINT Ø,LX,LY:GOTO 25Ø
24Ø IF PEEK(653)=1 THEN 5ØØ
25Ø DRAW CO TO X,Y
5ØØ BC=RDOT(X,Y)
51Ø POINT RND(1)*4,X,Y:
   FOR Z=1 TO 1Ø:NEXT Z
52Ø POINT CC(BC),X,Y:GOTO 1ØØ
```

When you RUN the program, you are asked to enter a mode in which to draw. You may enter Ø, 1, 2, 4 or 5. If you now press RETURN, the screen will clear and the words POINT and COLOUR 1, together with a flashing cross, will be printed. POINT is one of the ten drawing aids, which you can call up using the red User Defined Function (UDF) keys. Press each of the function keys, and notice that the display names the function. From fØ to f9, these are Point, Line, Triangle, Text, Ellipse, Draw, Colour change, Box, Frame and End.

The Point option lets you place a single point anywhere on the screen, but it's most important use is to establish the start of a line, box or rectangle, or the centre of a circle or ellipse. To draw a line on the screen, for example you would select Point (by pressing fØ), then use the arrow keys to move the cursor (flashing cross) to where you wish the line to start. Press the space bar to place a point on the screen. If you do not specify the start of the line with a point, then the last point visited by the cursor is taken as the start. You are now ready to draw the line. Press f1 to select Line, move the cursor to where you wish the line to end, then press the space bar, when the line will appear. This option lets you draw straight lines in any direction, but if you wish to draw freehand, as with a pencil, then you need to select Draw (f5).

When you select the Draw option, you can draw a line or curve in any direction, by holding down the space bar as you press any of the arrow keys. To stop drawing, release the space bar. At this stage you cannot change the drawing colour, which automatically defaults to red. The colour number is displayed at top right. When PROCCOLOUR is defined in part two, you will be able to alter this at will.

This program lets you save an image on the screen to tape or disk. In both cases, the file name 'PIC' is assigned automatically. If you use a disk system and your disk already has a file called PIC then it will be overwritten. To prevent this, you need to change the program and assign a file name instead of PIC at Lines 127Ø and 13ØØ. If you use a tape system, you can save more than one image with the same name (PIC), so there is no need to change it.

```
1Ø MODE6
15 *OPT1,Ø
2Ø PRINTTAB(4,5)"WELCOME
   TO INPUT'S C.A.D. PROGRAM"
   TAB(4,6)STRING$(33,CHR$(95))'''
3Ø INPUT"WHICH MODE WOULD YOU
   LIKE□",M
4Ø M=INT(M):IF M<Ø OR M>7 THEN 3Ø
5Ø IF M=6 OR M=7 OR M=3 THEN
   PRINT''"I'M VERY SORRY BUT THIS
   PROGRAM USES□□□GRAPHICS AND
   DOES NOT WORK IN THAT MODE":
   PRINT''"NOW□";:GOTO 3Ø
6Ø MODEM
7Ø IF M=Ø OR M=4 THEN MC=1
8Ø IF M=1 OR M=5 THEN MC=3
9Ø IF M=2 THEN MC=15
1ØØ *FX225,13Ø
11Ø *FX4,2
12Ø *FX226,14Ø
13Ø *FX227,14Ø
14Ø *FX228,14Ø
15Ø DIM A$(1Ø)
16Ø FOR T=1 TO 1Ø:READ A$(T):NEXT
17Ø DATA"POINT□□□□□□□□□□",
   "LINE□□□□□□□□□□□",
   "TRIANGLE□□□□□□",
   "TEXT□□□□□□□□□□□",
   "ELLIPSE□□□□□□□□",
   "DRAW□□□□□□□□□□□",
   "CHANGE COLOUR",
   "BOX□□□□□□□□□□□□",
   "FRAME□□□□□□□□□□",
   "END□□□□□□□□□□□□"
18Ø X=68Ø:Y=512:X2=X:Y2=Y:
   X3=X:Y3=Y:C=1
19Ø A$=CHR$(13Ø):PRINTTAB(Ø,Ø)
   A$(1)TAB(2Ø)"COLOUR□";C
2ØØ GCOLØ,C
21Ø REPEAT
22Ø PROCMOVE
23Ø A2$=INKEY$(1)
235 IF A2$=CHR$(19) THEN PROCSAVE
236 IF A2$=CHR$(12) THEN PROCLOAD
24Ø IF ASC(A2$)>129 AND ASC(A2$)
   <14Ø THEN A3$=A$:A$=A2$:PRINT
   TAB(Ø,Ø)A$(ASC(A2$)-129)
25Ø IF A$=CHR$(13Ø) THEN PROCPOINT
26Ø IF A$=CHR$(131) THEN PROCLINE
27Ø IF A$=CHR$(132) THEN PROCTRI
28Ø IF A$=CHR$(133) THEN
   PROCTEXT:GOTO 24Ø
29Ø IF A$=CHR$(134) THEN PROCELLIPSE
3ØØ IF A$=CHR$(135) THEN PROCDRAW
31Ø IF A$=CHR$(136) THEN
   PROCCOLOUR:GOTO 24Ø
32Ø IF A$=CHR$(137) THEN PROCBOX
33Ø IF A$=CHR$(138) THEN PROCFRAME
34Ø UNTIL A$=CHR$(139)
35Ø *FX4,Ø
36Ø END
37Ø DEF PROCLINE
38Ø IF NOT INKEY -99 THEN ENDPROC
39Ø MOVE X2,Y2:DRAW X,Y
4ØØ X3=X2:X2=X:Y3=Y2:Y2=Y
41Ø IF INKEY -99 THEN 41Ø
42Ø ENDPROC
43Ø DEF PROCMOVE
44Ø DX=Ø:DY=Ø
45Ø IF INKEY -58 THEN DY=4
46Ø IF INKEY -42 THEN DY=-4
47Ø IF INKEY -26 THEN DX=-2
48Ø IF INKEY -122 THEN DX=2
49Ø IF INKEY -1 THEN DY=DY*4:
   DX=DX*4
5ØØ IF INKEY -2 THEN DY=DY*2:
   DX=DX*2
51Ø X=X+DX:Y=Y+DY
52Ø PROCCUR
53Ø ENDPROC
54Ø DEF PROCCUR
55Ø MOVE X,Y+1Ø:PLOT 6,X,Y-1Ø:
   MOVE X+1Ø,Y:PLOT 6,X-1Ø,Y
56Ø MOVE X,Y+1Ø:PLOT 6,X,Y-1Ø:
   MOVE X+1Ø,Y:PLOT 6,X-1Ø,Y
57Ø MOVE X,Y
58Ø ENDPROC
59Ø DEF PROCPOINT
6ØØ IF NOT INKEY-99 THEN ENDPROC
61Ø PLOT69,X,Y
62Ø X3=X2:X2=X:Y3=Y2:Y2=Y
63Ø IF INKEY-99 THEN 63Ø
64Ø ENDPROC
65Ø DEF PROCTRI
```

```
710 ENDPROC
720 DEF PROCTEXT
730 A2$=A3$
820 ENDPROC
830 DEF PROCELLIPSE
910 ENDPROC
920 DEF PROCDRAW
930 IF NOT INKEY-99 THEN ENDPROC
940 PLOT69,X,Y
950 X3=X:X2=X:Y3=Y:Y2=Y
960 ENDPROC
970 DEF PROCCOLOUR
1070 ENDPROC
1080 DEF PROCBOX
1130 ENDPROC
1140 DEF PROCFRAME
1190 ENDPROC
1200 DEF PROCLOAD
1220 ENDPROC
1250 DEF PROCSAVE
1310 ENDPROC
```

The program begins by displaying a sub-menu, which lets you select values for mode, screen type and screen colour. Use the arrow keys to move the cursor and the space bar to select the value.

After entering these values, you should then press ENTER to display the main menu. To access items on the main menu, place the cursor (using the arrow keys) beside the item number, which is to the left of the screen, then press the space bar to select the item or option. This action clears the menu, leaving a blank screen with a drawing cursor. You can return to the main menu at any stage, simply by pressing ENTER.

The first option on the main menu is Change screen, which, if selected, takes you back to the sub-menu. The first of the drawing options is Draw, which you would use to draw freehand, as with a pencil. When you select this option, you can move the cursor to the point where you wish to start drawing, but you will begin to mark the screen only if you hold down the space bar while pressing one of the arrow keys. To stop drawing, merely release the space bar, then you can move to another point and hold down the space bar to start drawing again. You can speed up the movement of the cursor by holding down SHIFT on the dragon or CLEAR on the Tandy, but this works only if you are moving and not drawing.

In any of the drawing options, you can change the draw colour by pressing a key between 0 and 8; the colour is as specified in your User's Manual. To exit Draw, press ENTER, when the display will return to the main menu.

The Line option lets you draw straight lines. Select it with the arrow keys, move to where you wish to start the line, then press the space bar. You can now select a colour for the line (using keys 0 to 8). (The colour of the cursor itself depends on the colour of the screen beneath its centre but this does not affect the colour of the line.) Move to where you wish to end the line, then press the space bar to draw the line. Pressing ENTER gets you back to the main menu again. Use CLEAR to exit the program. (Note the changes for the Tandy at the end.)

```
10 PCLEAR8
20 CLS:PA=254:PB=253:PC=223:
   PD=191:PE=159
30 DIM SC(1228),CP(614)
40 DEFFNR(Z)=SQR((XS-X)*
   (XS-X)+(YS-Y)*(YS-Y))
50 MD=-1:ST=-1:OP=1:CL=1:
   X=127:Y=95
60 GOTO 80
70 FORK=1TO1500:NEXT
80 GOSUB 1000
```



```
90 IFMD<Ø THENPRINT@449,"error
   GRAPHICS MODE UNDEFINED":
   GOTO7Ø
1ØØ IFST<Ø THENPRINT@449,"error
   SCREEN TYPE UNDEFINED":
   GOTO7Ø
11Ø OT=1:PMODEMD,1
12Ø CLS:PRINT@11,"main menu"
13Ø PRINT@1Ø3,"1 – CHANGE
   SCREEN":PRINT@135,"2 –
   DRAW":PRINT@167,"3 –
   LINE":PRINT@199,"4 –
   RECTANGLE":PRINT@231,"5 –
   BOX":PRINT@263,"6 – CIRCLE"
14Ø PRINT@295,"7 – DISC":PRINT
   @327,"8 – ELLIPSE":PRINT@359,
   "9 – COPY":PRINT@39Ø,"1Ø –
   FILL":PRINT@422,"11 – ERROR":
   PRINT@454,"12 – SAVE/LOAD"
15Ø POKE1Ø97+OT*32,128
16Ø IFPEEK(341)=PC ANDOT>1
   THENPOKE1Ø97+OT*32,96:
   OT=OT–1:GOTO15Ø
17Ø IFPEEK(342)=PC ANDOT<12
   THENPOKE1Ø97+OT*32,96:
   OT=OT+1:GOTO15Ø
18Ø IFPEEK(345)=PC THEN2ØØ
19Ø IFPEEK(339)=PD THENCLS:
   END ELSE16Ø
2ØØ IFOT<>11 THEN22Ø
21Ø PMODEMD,5:PUT(Ø,Ø)–(255,
   191),SC:GOSUB5ØØ:PMODE
   MD,1:GOTO12Ø
22Ø GOSUB51Ø
23Ø GET(Ø,Ø)–(255,191),SC
24Ø EF=Ø
```

Is there an easy way to remember which User Defined Function key gives each of the drawing aids on the BBC and Electron micros?

The ten function keys on these micros are used extensively, so a means of ready reference would avoid confusion while you learn to use the program. The transparent bar above the red keys of the BBC micro is intended for just this use. Cut a strip of paper to fit under the bar, and divide it into key width boxes. Write each function—such as Draw and Circle—in its box and slide the strip under the bar. Electron users can merely stick the strip into place with adhesive tape.

```
25Ø ON OT GOSUB1ØØØ,2ØØØ,3ØØØ,
   3ØØØ,3ØØØ,3ØØØ,3ØØØ,3ØØØ,
   4ØØØ,5ØØØ,Ø,6ØØØ
26Ø GOTO12Ø
5ØØ FORK=1TO4:PCOPYK+4TOK:
   NEXT:RETURN
51Ø FORK=5TO8:PCOPYK–4TOK:
   NEXT:RETURN
1ØØØ CLS:PRINT@8,"screen set-up"
1Ø1Ø PRINT@1Ø2,"1 – GRAPHICS
   MODE":PRINT@166,"2 – SCREEN
   TYPE":PRINT@23Ø,"3 – CLEAR
   SCREEN"
1Ø2Ø PK=PEEK(1Ø62+64*OP):
   POKE 1Ø62+64*OP,63ANDPK
1Ø3Ø IFPEEK(341)=PC ANDOP>1
   THENPOKE1Ø62+64*OP,PK:
   OP=OP–1:GOTO1Ø2Ø
1Ø4Ø IFPEEK(342)=PC ANDOP<3
   THENPOKE1Ø62+64*OP,PK:
   OP=OP+1:GOTO1Ø2Ø
1Ø5Ø IFPEEK(338)=PD THENRETURN
1Ø6Ø IFPEEK(345)=PC THEN1Ø8Ø
1Ø7Ø GOTO1Ø3Ø
1Ø8Ø CLS:ON OP GOSUB12ØØ,13ØØ,14ØØ
1Ø9Ø FORK=1TO2ØØ:NEXT:GOTO1ØØØ
12ØØ PRINT@33,"WHICH GRAPHICS MODE
   DO YOU WISH TO USE (Ø–4) ?";
121Ø A$=INKEY$:IFA$<"Ø"OR
   A$>"4"THEN121Ø
122Ø PRINTA$:MD=VAL(A$):
   PMODE MD,1:RETURN
13ØØ IF MD<ØTHENPRINT@449,"error
   GRAPHICS MODE UNDEFINED":
   FORK=1TO1ØØØ:NEXT:RETURN
131Ø PRINT@33,"WHAT SCREEN TYPE DO
   YOU WISH TO USE (Ø OR 1) ?";
132Ø A$=INKEY$:IFA$<"Ø"OR
   A$>"1"THEN132Ø
133Ø PRINTA$:ST=VAL(A$):RETURN
14ØØ PRINT@33,"ARE YOU SURE YOU WISH
   TO CLEAR□□THE SCREEN (Y/N) ?"
141Ø A$=INKEY$:IFA$<>"Y"AND
   A$<>"N"THEN141Ø
142Ø IFA$="N"THENRETURN
143Ø PRINT@129,"WHICH COLOUR DO YOU
   WISH TO□□□□□CLEAR THE SCREEN
   WITH (Ø–8) ?";
144Ø A$=INKEY$:IFA$<"Ø"OR
   A$>"8"THEN144Ø
145Ø PRINTA$:PMODEMD,5:PCLS
   VAL(A$):PMODEMD,1:PCLS
   VAL(A$):RETURN
15ØØ FORK=ØTO7:IFPEEK(338+K)
   =PA THENCL=K
151Ø NEXT:IFPEEK(338)=PB THENCL=8
152Ø IFPEEK(338)=PD THEN
   EF=1:RETURN
153Ø DRAW"BM"+STR$(X)+","+
   STR$(Y)+";C"+STR$((PPOINT
   (X,Y)+3)AND7)+"BE4G2BD4NF2BL4
   NG2BU4H2":COLORCL
154Ø IFPEEK(337)=255THEN15ØØ
155Ø IFPEEK(337)=PE THEN
   DF=1Ø ELSEDF=1
156Ø IFPEEK(341)=PC THEN
   Y=Y–DF:GOTO161Ø
157Ø IFPEEK(342)=PC THEN
   Y=Y+DF:GOTO161Ø
158Ø IFPEEK(343)=PC THEN
   X=X–DF:GOTO161Ø
159Ø IFPEEK(344)=PC THEN
   X=X+DF:GOTO161Ø
16ØØ RETURN
161Ø X=255ANDX
162Ø IFY>191ORY<Ø THEN
   Y=Y+191*(2*(Y>191)+1)
163Ø GOSUB5ØØ:RETURN
2ØØØ SCREEN1,ST:GOSUB15ØØ
2Ø1Ø IFEF=1 GOSUB5ØØ:RETURN
2Ø2Ø IFPEEK(345)=PC THEN
   PMODEMD,5:PSET(X,Y,CL):
   PMODEMD,1:POKE337,255
2Ø3Ø GOTO2ØØØ
3ØØØ SCREEN1,ST:GOSUB15ØØ
3Ø1Ø IFEF=1 GOSUB5ØØ:RETURN
3Ø2Ø IFPEEK(345)<>PC THEN3ØØØ
3Ø3Ø XS=X:YS=Y
3Ø4Ø GOSUB15ØØ
3Ø5Ø GOSUB5ØØ
3Ø6Ø IF EF=1 THEN RETURN
3Ø7Ø ON OT GOSUBØ,Ø,313Ø,314Ø,
   314Ø,315Ø,315Ø,316Ø,Ø,Ø,Ø
3Ø8Ø IFPEEK(345)=PC THEN31ØØ
3Ø9Ø GOTO 3Ø4Ø
31ØØ IF OT=5 THENLINE(X,Y)–
   (XS,YS),PSET,BF
311Ø IF OT=7 THENPAINT(X,Y),CL,CL
312Ø GOSUB51Ø:GOTO3ØØØ
313Ø LINE(XS,YS)–(X,Y),PSET:RETURN
314Ø LINE(XS,YS)–(X,Y),PSET,B:RETURN
315Ø CIRCLE(X,Y),FNR(Z),CL:RETURN
316Ø IF XS<>X THEN 319Ø
317Ø IF (2*YS–Y)>191 OR (2*YS–Y)<Ø
   THEN RETURN
318Ø LINE(X,Y)–(X,2*YS–Y),PSET:
   RETURN
319Ø CIRCLE(XS,YS),ABS(X–XS),
   CL,ABS((Y–YS)/(X–XS)):
   RETURN
4ØØØ RETURN
5ØØØ RETURN
6ØØØ RETURN
```

For the Tandy, delete Line 154Ø, change the 337 in Line 2Ø2Ø to 345 and alter these lines:

```
2Ø CLS:PA=239:PB=223:
   PC=247:PD=191:PE=183
155Ø IF PEEK(339)=PD THEN DF=1Ø
   ELSE DF=1
```

CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

COMING IN ISSUE 19...

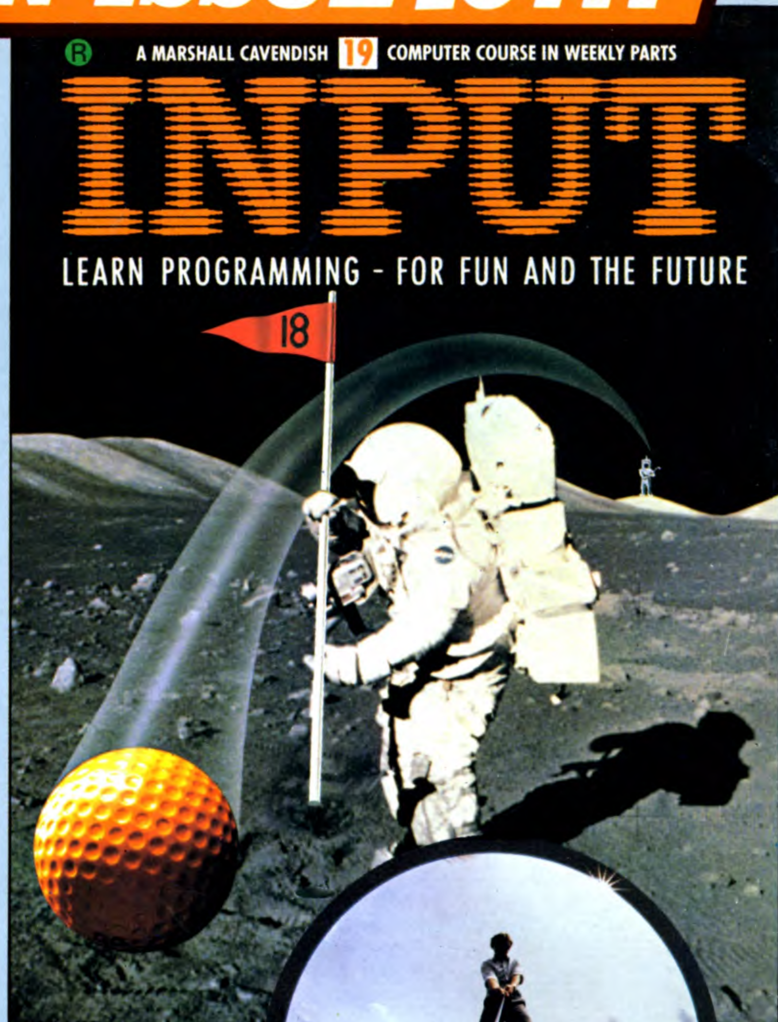
- *Learn how you can turn mathematical equations into vivid screen displays—and create a BOUNCING BALL*

- *If you need a function that isn't part of standard BASIC, you may be able to program in a special CUSTOM FUNCTION that will do the job for you*

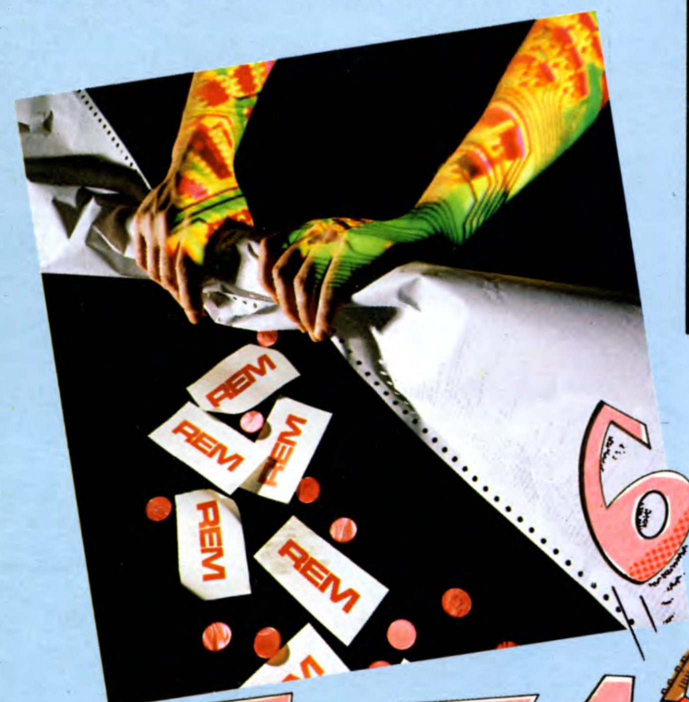
- *Complete your COMPUTER DRAWING PROGRAM by adding a series of more sophisticated routines*

- *Now you have had your turn at PONTOON, it's time to give the computer a go—and see who has won*

- *And for ACORN users, the PROGRAM SQUEEZER is completed by a routine that tells you how much you have saved*









ASK YOUR NEWSAGENT FOR INPUT